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

Lab Guide

Debugging 201:

Practical Troubleshooting Concepts & Tools

Josh Addington & Robert Ninness

Default Login / Password:

admin / Knowledge15

itil / Knowledge15

employee / Knowledge15

This

Page

Intentionally

Left

Blank

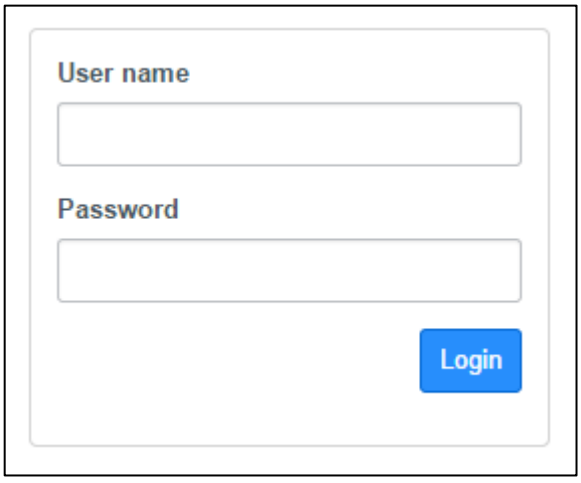
Lab Goal

ServiceNow provides a variety of Debug options. This lab explains how to enable and view output from the **Debug Business Rule** debug module to troubleshoot a rule that isn't firing.

Lab 1
Debug Business Rules

Lab Instance Log On

1. Follow on-screen instructions to request your unique instance URL.
2. Log on to the instance using the credentials provided, admin / Knowledge15.



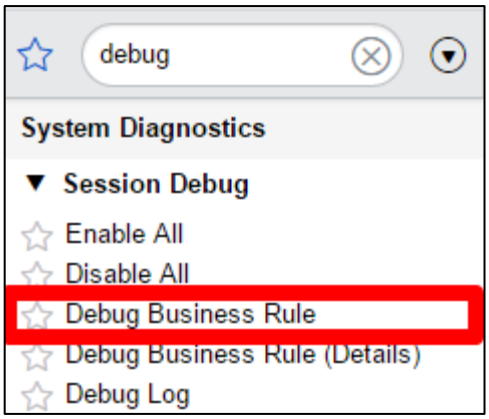
®2015 ServiceNow, Inc. All rights reserved.

Open an Existing Incident

1. Navigate to **Incident > Open**.
2. Click on any open incident record.
3. Set Incident State to **Closed**; and set **Close Code** and **Close Notes**.
4. Right click the header and **Save** the record.
5. **Result**: no comment or info message added.
6. **Question**: Did the Business Rule execute?

Enable Business Rule Debugging

1. Navigate to **System Diagnostics > Session Debug** (search for "debug").
2. Click **Debug Business Rule** (do not enable the option with "**(Details)**" yet; you do this later).

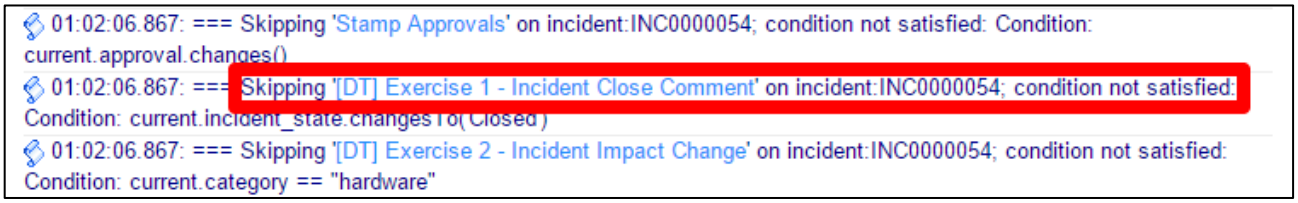


3. Repeat the steps from above in **Open an Existing Incident**.
4. Scroll down to see the output at the bottom of the screen and notice the following:

   Skipping '[DT] Exercise 1 - Incident Close Comment' on table incident; **condition not satisfied**: **current.incident_state.changesTo("Closed")**.

©2015 ServiceNow, Inc. All rights reserved.

5. From the debug output, **click on the business rule name** to open it.



Remove the Business Rule Condition

1. Click on **Business Rule name in blue** to navigate to the Exercise 1 Business Rule.
2. Click the **Advanced** tab.
3. Remove the condition.



4. Comment out the **current.comments** set statement.

```
8  // add comment when closing
9  //current.comments = "Closed by " + gs.getUserName() + " at " + gs.nowDateTime();
10 gs.addInfoMessage( "Close comment added" );
```

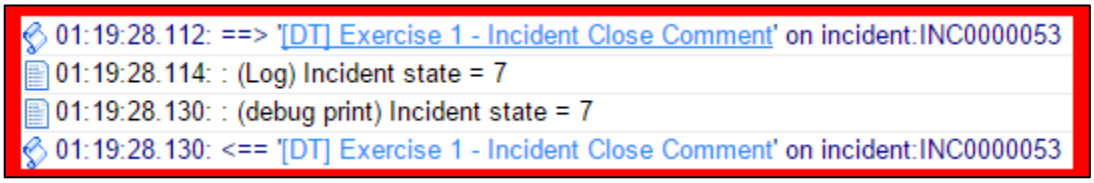
5. Uncomment the existing debugging statements as shown.

```
12 // Debug info
13 gs.addInfoMessage("(InfoMessage) Incident state = " + current.incident_state); //output to app
14 gs.log("(Log) Incident state = " + current.incident_state);                    //output to system log
15 gs.print("(debug print) Incident state = " + current.incident_state);          //output to debug console
```

6. Update the **Business Rule**.
7. Navigate to another open incident.
8. Set the **State** to **Closed**; set **Close code** and **Close notes**.
9. Save the incident.

©2015 ServiceNow, Inc. All rights reserved.

10. Scroll down to view the debug output and note the log statements.



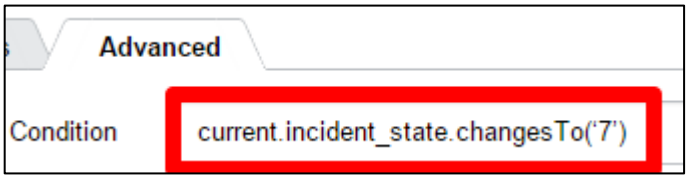
11. It looks like the Business Rule condition has the wrong value!

©2015 ServiceNow, Inc. All rights reserved.

Fix the Business Rule

1. Navigate to the **Business Rule** by clicking on the name in **blue** text.
2. Add the correct the **condition**, shown below.

   `current.incident_state.changesTo('7')`



3. Next, update the Script portion of the Business Rule as follows.

   Uncomment the **current.comments** set statement on line 9 as shown.

   Comment out lines 13, 14 and 15 from the existing debugging statements as shown.

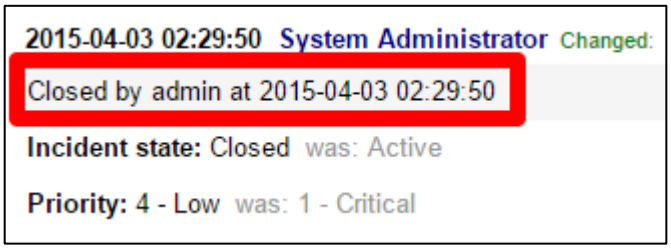
```
9  current.comments = "Closed by " + gs.getUserName() + " at " + gs.nowDateTime();
11
12 // Debug info
13 //gs.addInfoMessage("(InfoMessage) Incident state = " + current.incident_state); //output to app
14 //gs.log("(Log) Incident state = " + current.incident_state);                    //output to system log
15 //gs.print("(debug print) Incident state = " + current.incident_state);          //output to debug console
```

4. Update the Business Rule.

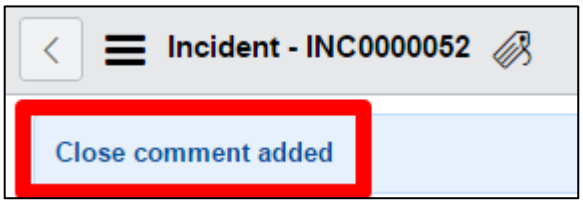
©2015 ServiceNow, Inc. All rights reserved.

Lab Success Verification

1. Navigate to another open incident.
2. Set the **State** to **Closed**.
3. Set **Close code** and **Close notes**.
4. Save the Incident.
5. Verify that the comment was correctly added.



6. Verify that the info message was displayed.



®2015 ServiceNow, Inc. All rights reserved.

Lab Goal

Lab 2 Debug Business Rules (Details)

This lab explains how to use **Debug Business Rules (Details)** to determine why the Impact is changing on newly created Incidents.

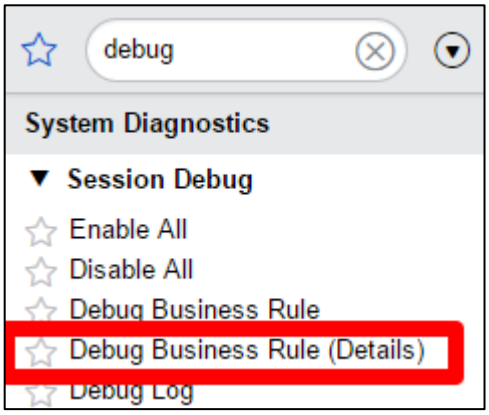
Background: Whenever a hardware incident is created, the impact is set to 1. We suspect it might be due to a business rule.

Try it

1. Create a **new incident**; set the category to **Hardware** and save it.
2. Notice that the impact value has changed to **1**.

Enable Business Rule Debugging (Details)

1. Navigate to **System Diagnostics > Session Debug**.
2. Click **Debug Business Rule (Details)**.



3. Create another **hardware incident** and save it.
4. Search the debug output for "**impact**". This output shows us that a business rule is indeed changing the impact and gives us the name of the business rule responsible.



©2015 ServiceNow, Inc. All rights reserved.

Lab Goal

Lab 3
Debug Log
Try / Catch

This lab explains how to use **try/catch** in JavaScript to handle errors. It also explains how to use debug log to view **try/catch** output.

Scenario: A simple business rule we have written should display an info message when we save a software incident. However, we are only getting one of the three messages we expect.

Try it

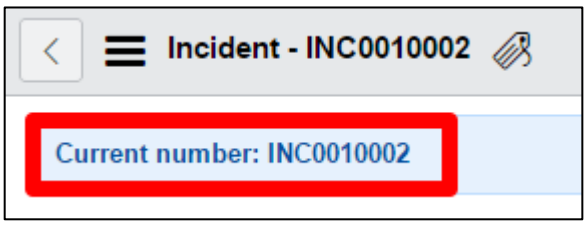
1. Take a look at Business Rule **[DT] Exercise 3 - Display Info.** This business rule only executes when the Incident **Category** is **Software** and the record is inserted or updated. It should add three info messages to the top of the form when executed.

```
9  //output current number
10 gs.addInfoMessage("Current number: " + current.number);
11
12 //output current short_description
13 gs.addInfoMessage("Current Short Description: " + currrent.short_description);
14
15 //output current category
16 gs.addInfoMessage("Current Category: " + current.category);
```

2. Create an Incident with **Software category.**

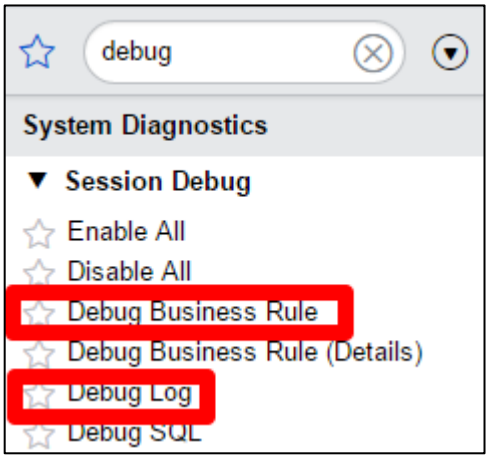
3. **Add a short description** and save the incident.

4. **Result:** Notice one **message** at the top of the form but not the other two.

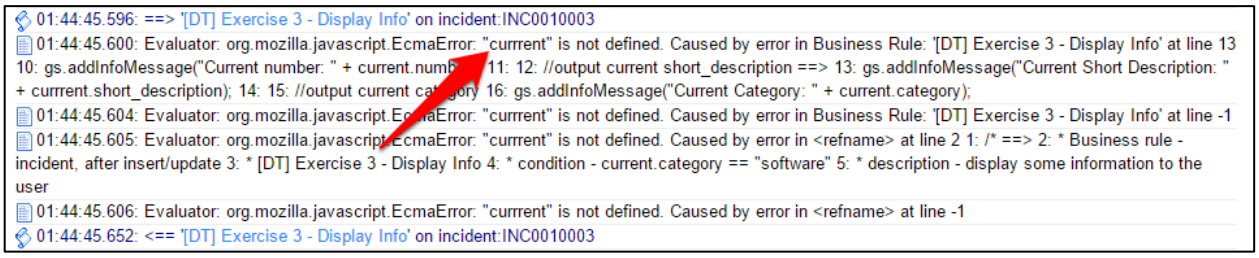


©2015 ServiceNow, Inc. All rights reserved.

5. Enable both **Debug Business Rule** and **Debug Log**.



6. **Create another software incident** and view **the debug** output. Notice there is a script exception because there is an extra "**r**" in "current" (do not fix this yet).



©2015 ServiceNow, Inc. All rights reserved.

Adding a try … catch block

1. From the debug output, **click** the business rule to open it.

2. Add a **try catch block** around the statements and save the business **rule**. Notice that there is one statement left below the catch block.

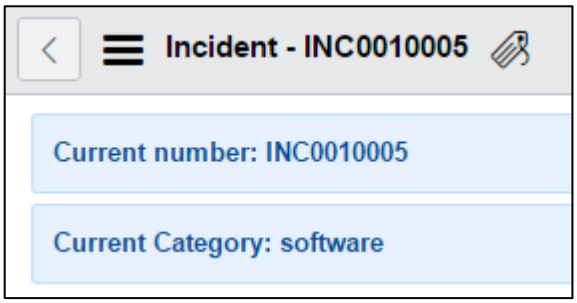
```
try{
//output current number
gs.addInfoMessage("Current number: " + current.number);

//output current short_description
gs.addInfoMessage("Current Short Description: " + currrent.short_description);
}

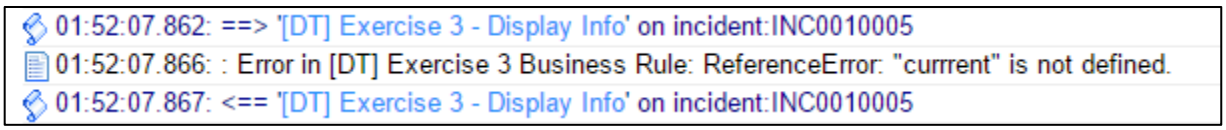
catch(err) {
    gs.log("Error in [DT] Exercise 3 Business Rule: " + err);
}

//output current category
gs.addInfoMessage("Current Category: " + current.category);
```

3. Create one more software **Incident** and save it. Note, now there are two info messages!



4. Examine the debug output and notice how the catch statement provides more concise information about the error. Catching this error allows the code following the error in the business rule to execute, which enables the second info message to appear.



5. Disable **All** debugging tools.

©2015 ServiceNow, Inc. All rights reserved.

Summary

Unhandled errors can prevent script functionality and even other scripts from running. Catch statements guarantee that **you** control what happens when an error occurs. This is especially important if you need to clean up or revert something due to the original error.

©2015 ServiceNow, Inc. All rights reserved.

Lab Goal

This lab explains how to use the JavaScript debugger console.

Lab 4
Client Script Debugger

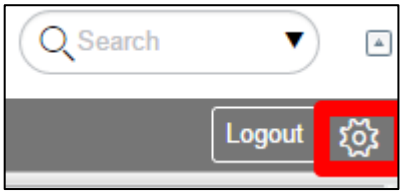
Enable the JavaScript Debugger Console

1. The Incident table has an **onChange** Client Script on the category field called **[DT] Exercise 4 - Request Category.**

```
function onChange(control, oldValue, newValue, isLoading, isTemplate) {
    if (isLoading || newValue == '')
        return;

    if (newValue == "request") {
        g_form.setValue("urgency", 3);
        jslog("[DT] Exercise 4 Request Category client script: Setting urgency to low for requests");
    }
}
```

2. Enable the Client JavaScript Debugger Console by clicking the cog in the top-right corner of the screen and then clicking **JavaScript Log and Field Watcher**.





©2015 ServiceNow, Inc. All rights reserved.

3. Create a new **Incident**. Set **Urgency** to **2 – Medium**.
4. Change the **Category** to **Request**.
5. This causes the onChange script to fire. Notice the **jslog** statement in the Debugger output.
6. Review debug console.



©2015 ServiceNow, Inc. All rights reserved.

Lab Goal

This lab explains how to use AJAX callbacks to optimize client script performance.

Scenario: When the caller on an Incident changes, we want to highlight VIPs. We notice that the browser hangs briefly when we do this.

Lab 5 Client Script Performance

View Client Script

1. Navigate to **System Definition > Client Scripts**.
2. Open **[DT] Exercise 5 – Highlight VIP Caller**.
3. Take note of this line:

```
var caller = g_form.getReference('caller_id');
```

Try it

1. Create a new **Incident**.
2. Set caller to **Fredy Luddy**.
3. Set caller to **Jerrod Bennett** – bingo!



Improve the Script

What's wrong with this the Client Script? **([DT] Exercise 5 – Highlight VIP Caller).**

Answer: It is making a **BLOCKING** call to the server.

©2015 ServiceNow, Inc. All rights reserved.

4. Open Client Script [DT] Exercise 5 – Highlight VIP Caller.

   Change code starting at line **19**.

   Before:

```
19      var caller = g_form.getReference('caller_id');
```

   After: Anonymous callback function. Wrap the subsequent code in braces as shown.

```
19      var caller = g_form.getReference('caller_id', function(caller) {
20
21      var callerLabel = $('label.incident.caller_id');
22      var callerField = $('sys_display.incident.caller_id');
23      if (!callerLabel || !callerField)
24          return;
25
26      //check for VIP status
27      if (caller.vip == 'true') {
28          var bgPosition = "95% 55%";
29          if (document.documentElement.getAttribute('data-doctype') == 'true')
30              bgPosition = "5% 45%";
31
32          callerLabel.setStyle({backgroundImage: "url(images/icons/vip.gif)", backgroundRepeat: "no-repeat", backgroundPosition: bgPosition});
33          callerField.setStyle({color: "red"});
34      } else {
35          callerLabel.setStyle({backgroundImage: ""});
36          callerField.setStyle({color: ""});
37      }
38  });
```

5. What did you do? You turned the blocking call into a nonblocking call with a callback. This example used an **anonymous** function.

6. Another way to get the same result is to create a **named** function. This is achieved by using code similar to the following example:

```
1   var caller = g_form.getReference('caller_id', vipCallback);
2
3   function vipCallback(caller) {
4       //Your Code Here!!!
5   }
```

®2015 ServiceNow, Inc. All rights reserved.

Lab Goal

This lab explains how to use **Display Business Rules** and **g_scratchpad** to improve performance.

Lab 6 Display Business Rules

Scenario: A client script needs to access data from a field that is not on the form.

Options:
- Add the field to the form view and hide it using UI Policy.
- Use a client GlideRecord query using Ajax.
- Use a display business rule to preload data and pass it to the client (**best**).

Review Business Rule and Client Script

1. Review Business Rule **[DT] Exercise 6 – Send data to client** on the Problem table.

```
/*
 * Business rule - problem, display
 * [DT] Exercise 6 - Send data to the client
 * Desciption - send data to client for use in client script [DT] Exercise 6 - Notify on old problems
 */

// display business rule on problem
// get the record sys_created_by value and add to the client scratchpad
g_scratchpad.created_by = current.sys_created_by;
gs.print("SCRATCH Createdby = " + g_scratchpad.created_by);

// calculate the age of this record
var age = gs.dateDiff(current.sys_created_on, gs.nowDateTime(), true);
var daysOld = Math.round(age / 3600 / 24);
g_scratchpad.days_old = daysOld;
gs.print("SCRATCH age = " + g_scratchpad.days_old);
```

©2015 ServiceNow, Inc. All rights reserved.

2. Review Client Script **[DT] Exercise 6 - Notify on old problems** on Problem table.

```
/*
 * Client Script - problem, onLoad
 * [DT] Exercise 6 - Notify on old problems
 * Description - get data from display business rule and alert user on old problems
 */

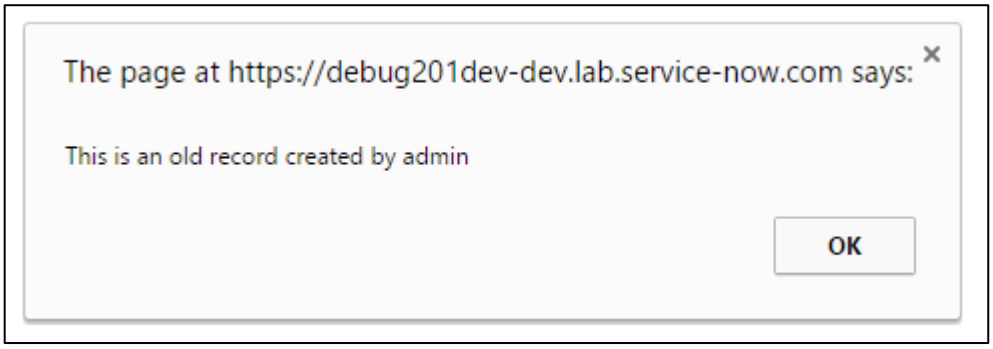
function onLoad() {
    // get data from display rule scratchpad
    var createdBy = g_scratchpad.created_by;
    var daysOld = g_scratchpad.days_old;

    if (parseInt(daysOld) > 30) {
        alert("This is an old record created by " + createdBy);
    }
}
```

Try it

1. Open the list of Problems and find one that was created more than 30 days ago.

2. Notice the alert from the client script.



Summary: Use display business rules and **g_scratchpad** to improve performance when sending data to the client. Avoid using GlideRecord in Client Scripts whenever possible.

©2015 ServiceNow, Inc. All rights reserved.

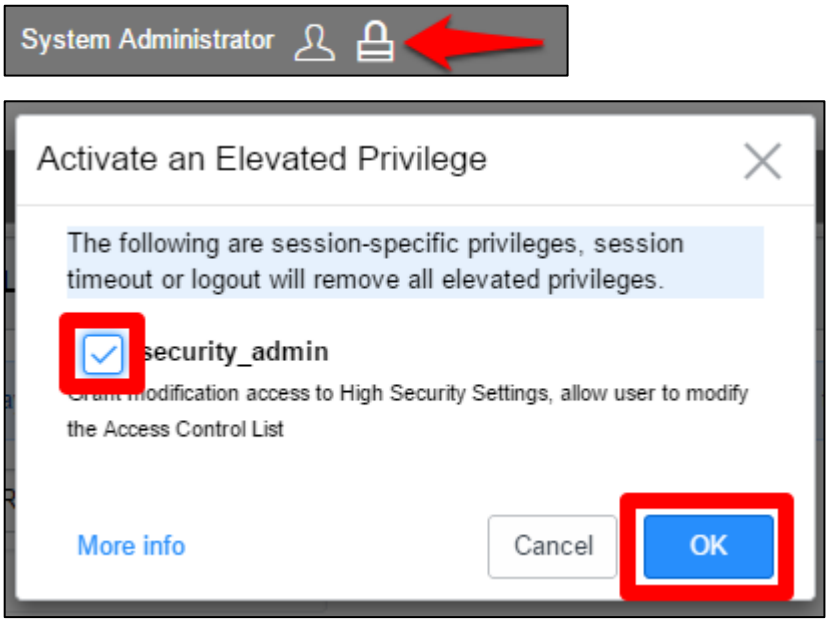
Lab Goal

This lab explains how to use **setLimit()** to reduce the number of records returned by a **GlideRecord** query.

Lab 7
GlideRecord
setLimit()

Without setLimit()

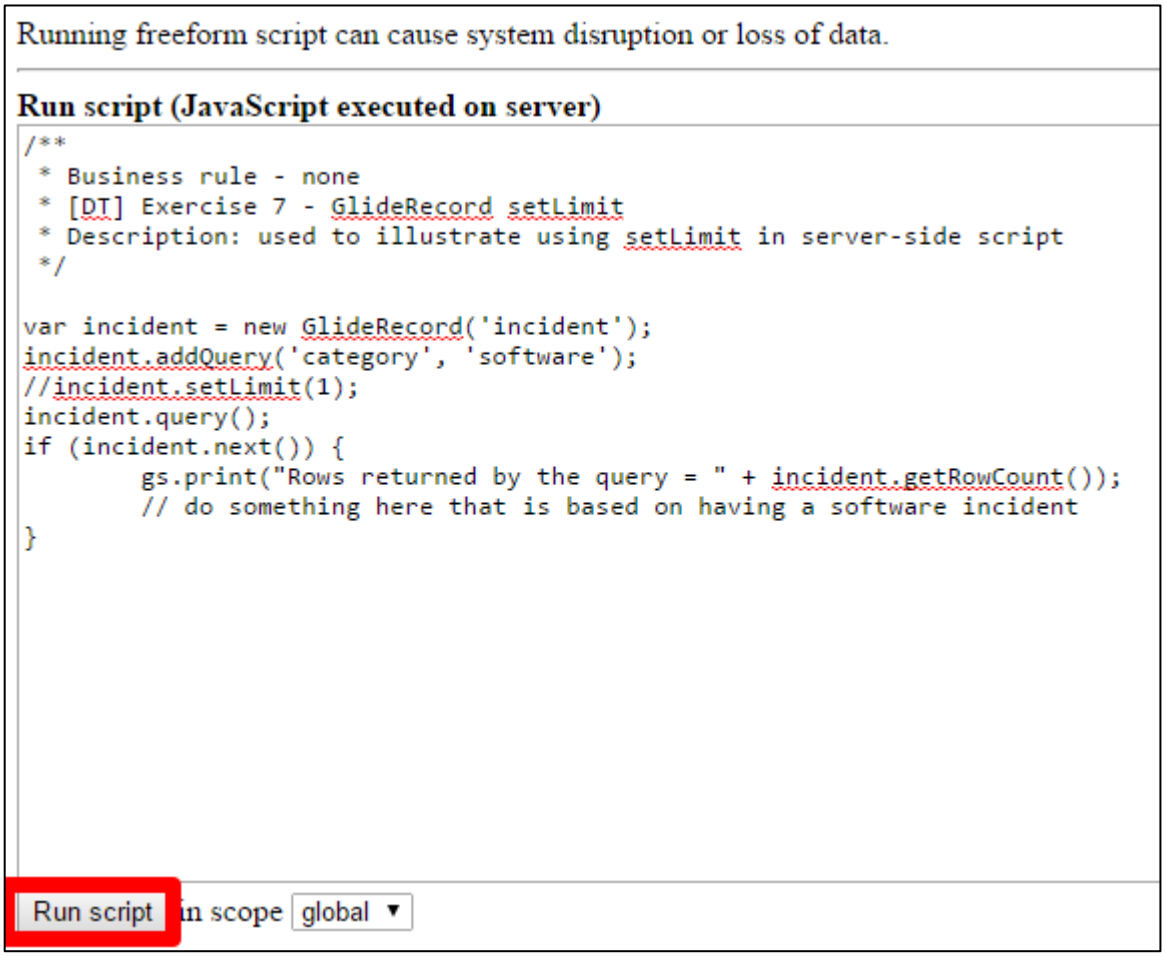
1. Navigate to **System Definition > Business Rules**.
2. Open **[DT] Exercise 7 - GlideRecord setLimit**.
3. Copy the contents of the Script field.
4. Activate the **security_admin** elevated privilege.



5. Navigate to **System Definition > Scripts – Background**.

©2015 ServiceNow, Inc. All rights reserved.

6. Paste the script and into the text area and click **Run script**.



7. Notice the output (this number varies, depending on how many software incidents are in the instance).

```
*** Script: Rows returned by the query = 14
```

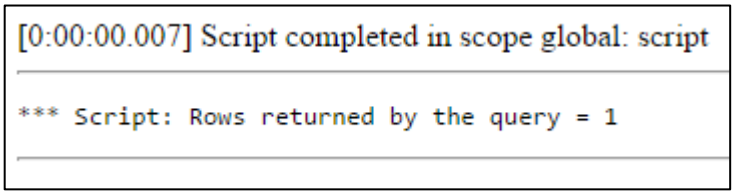
©2015 ServiceNow, Inc. All rights reserved.

A better way

The previous query gets slower and slower as the number of records in the system increases. To mitigate this, use **setLimit()**.

1. Return to **Scripts – Background**.
2. Paste the script again.
3. Uncomment this line:

   setLimit(1);
4. Run again.
5. Notice only one record returns this time. Much better!



Summary: Always remember to construct your queries so they return as little data as possible. If your query must return multiple records but will only be used to determine “is there a record?” consider using **setLimit(1)** to reduce the rows returned by the query.

®2015 ServiceNow, Inc. All rights reserved.

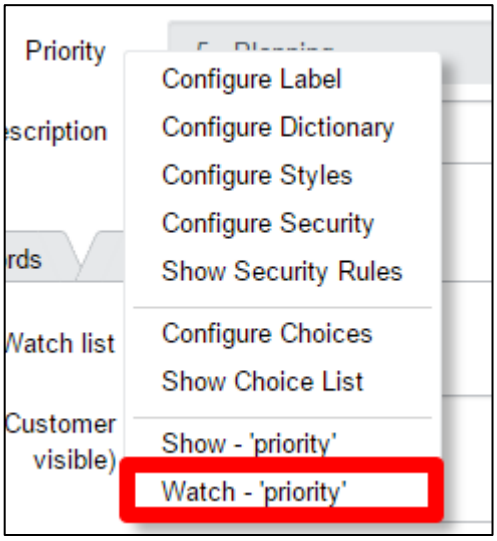
Lab Goal

This lab explains how to use the Field Watcher to determine why a field value is changing.

Lab 8
Field
Watcher

Enable Field Watcher

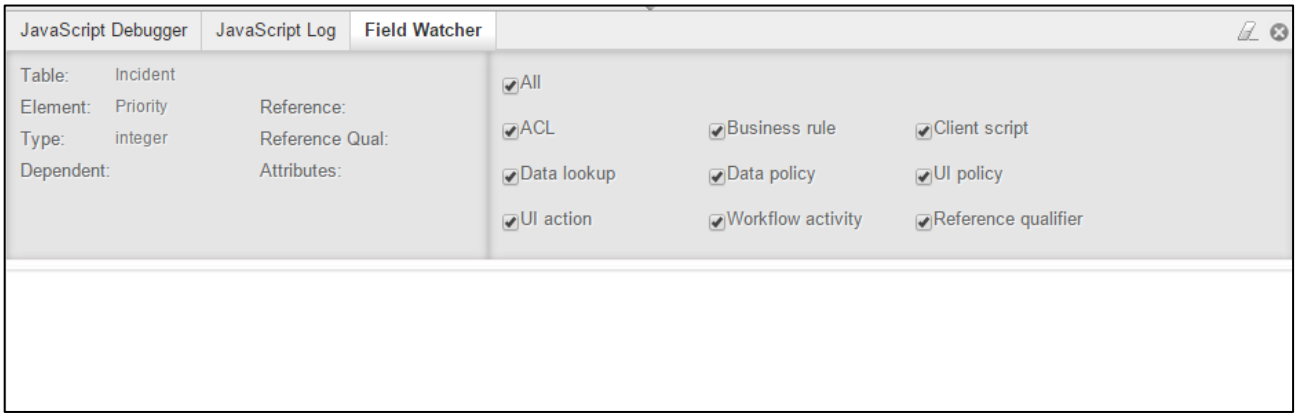
1. Open an existing Incident record.
2. Right-click the **Priority** label and click **Watch – 'priority'**.



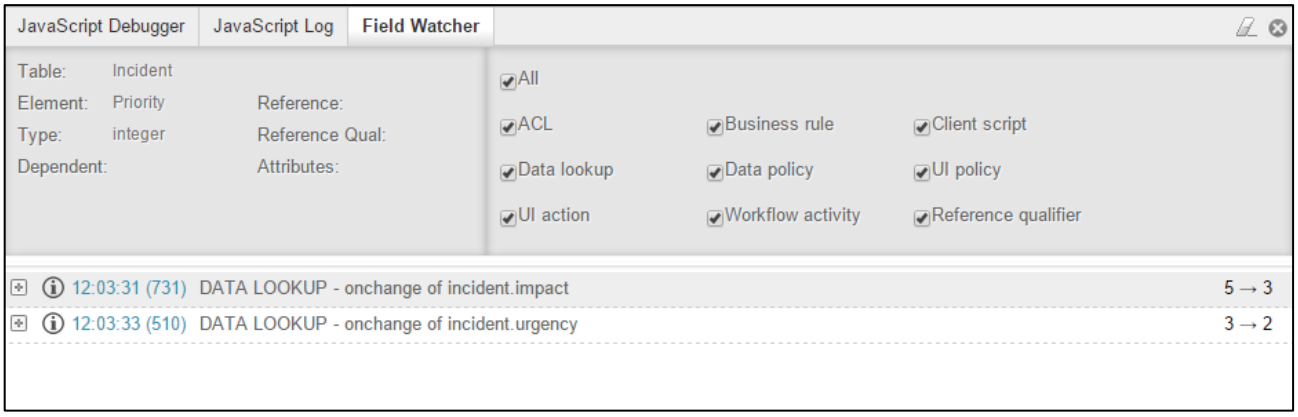
3. The field watcher appears at the bottom of the screen.

®2015 ServiceNow, Inc. All rights reserved.

4. Change the **Impact** and **Urgency** values on the Incident record and observe the output in the field watcher.



5. Notice that a data lookup rule is setting the Impact whenever Impact or Urgency change.



Disable Field Watcher

1. Right click the **Priority** label again, click **Unwatch – 'Priority'**.

2. Click the small **X** in the upper-right corner of the field watcher panel.



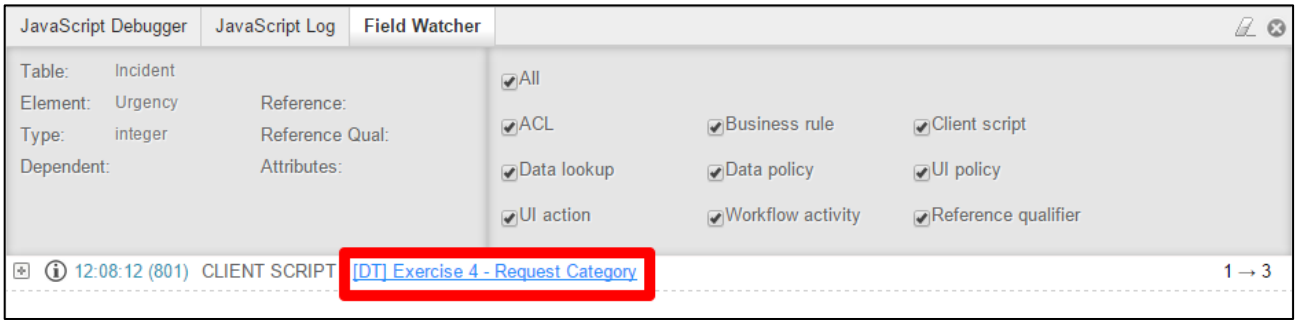
®2015 ServiceNow, Inc. All rights reserved.

Optional – Explore Some More

1. Create a new incident record, set the **Category** to **Software**; **Impact** to **1**; **Urgency** to **1**. **Save**.
2. Enable the Field Watcher for the **Urgency** field.
3. Clear the field watcher output by clicking the **eraser** icon.



4. Change the Category to Request.
5. Notice that the earlier client script is to blame for the urgency change. If desired, you can click on the script name to go directly to the Client Script.



®2015 ServiceNow, Inc. All rights reserved.

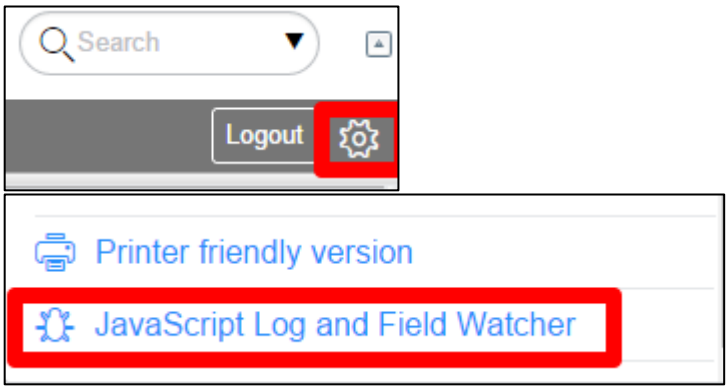
Lab Goal

This lab explains how to use the JavaScript debugger

Lab 9
JavaScript Debugger

Set a Breakpoint

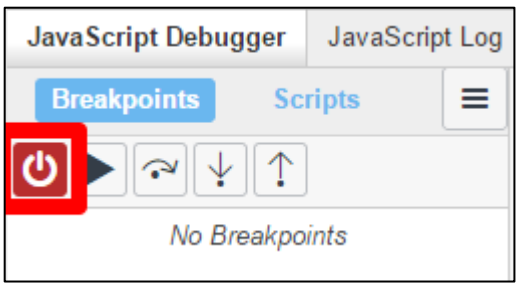
1. Enable the **JavaScript Debugger / Log / Watcher** pane.



2. If necessary, change to the **JavaScript Debugger** tab.

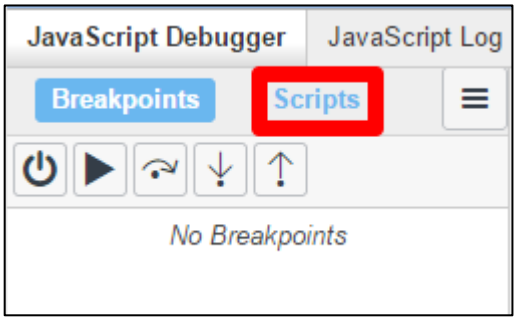


3. If the **Toggle debugger** button is **red**, click to disable it.

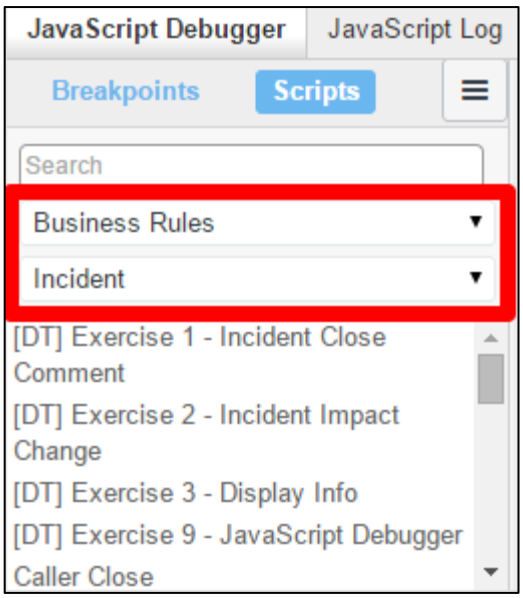


©2015 ServiceNow, Inc. All rights reserved.

4. Click **Scripts**.

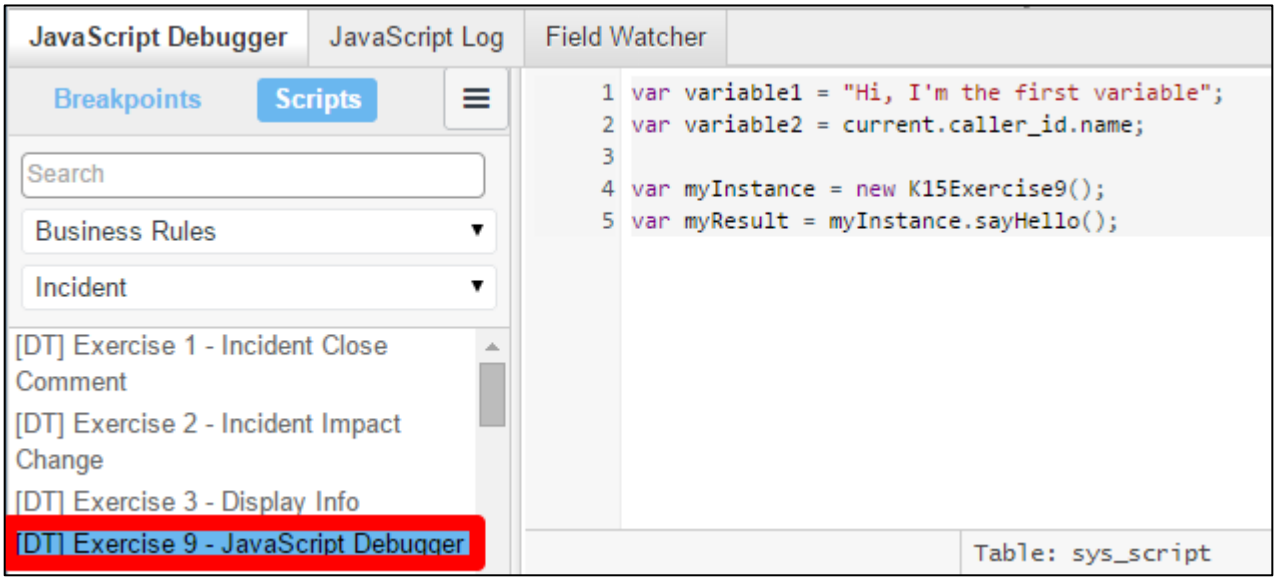


5. Change Type to **Business Rules** and Table to **Incident**.

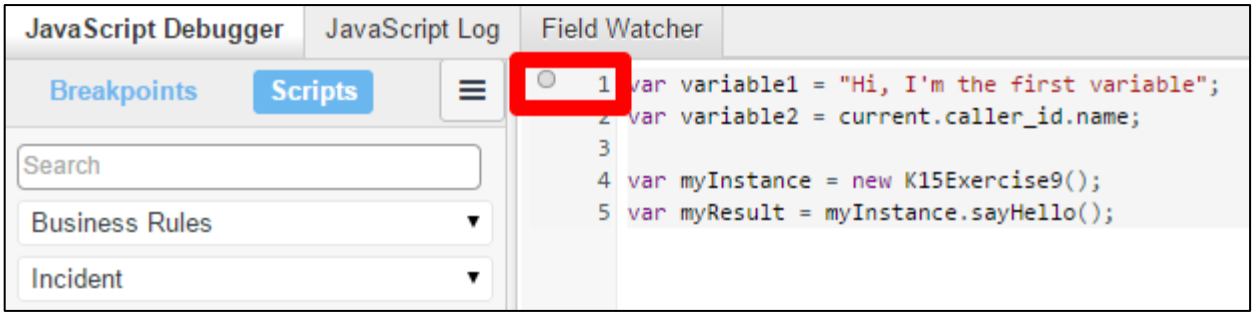


©2015 ServiceNow, Inc. All rights reserved.

6. Click **[DT] Exercise 9 – JavaScript Debugger** from the list of scripts.



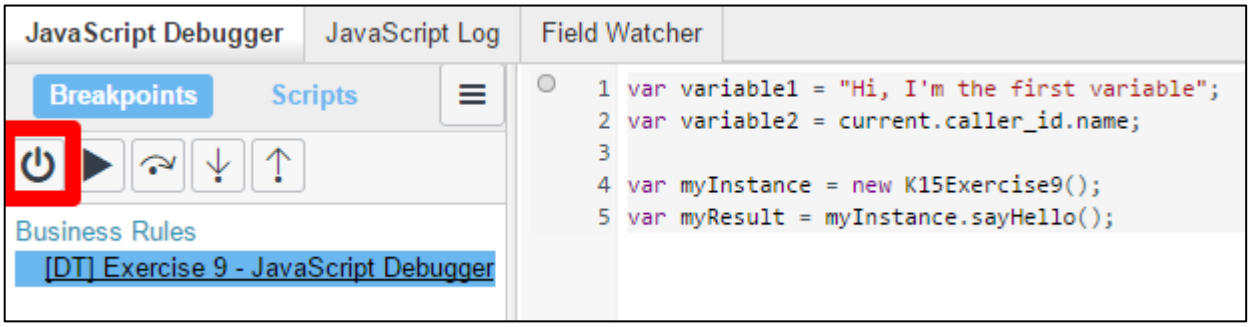
7. Set a breakpoint on **line 1** by clicking to the left of the line number.



8. Navigate to **Incident > Open**, and click on any **Incident** record.

©2015 ServiceNow, Inc. All rights reserved.

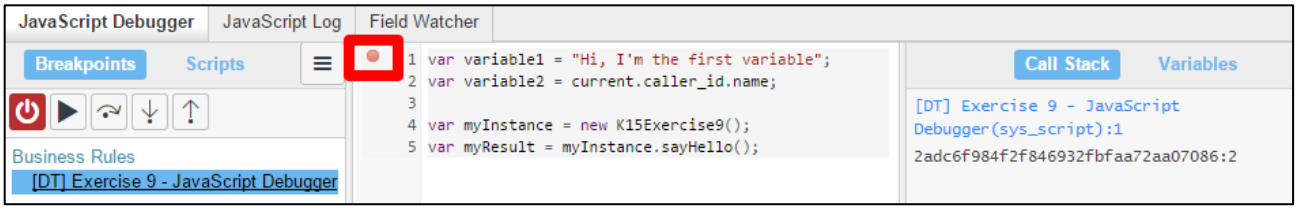
9. Change back to the **Breakpoints** subtab and **Enable** debugging. Notice the business rule now in the list. Any time you set a breakpoint, the related record shows up in the **Breakpoints** tab.



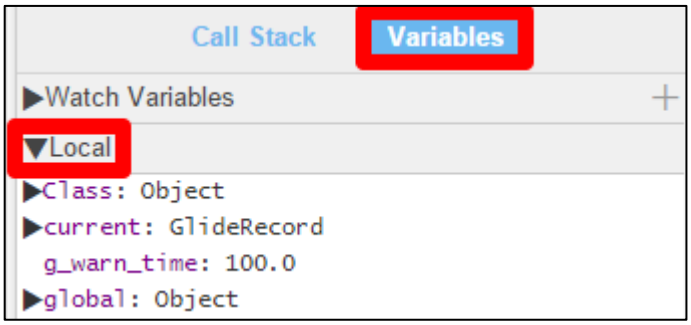
10. Change the Incident **Category** to **Request.**

11. Right click the header and **Save.**

12. The breakpoint indicator should turn **red**, and the page remains in a loading state. The **Call Stack** indicates where you stopped.

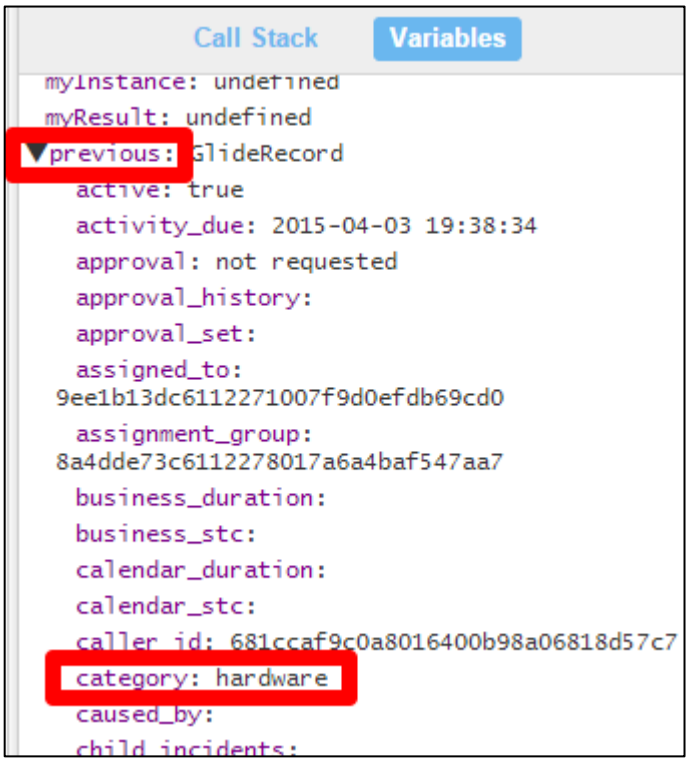


13. Change to the **Variables** tab and expand **Local**.



©2015 ServiceNow, Inc. All rights reserved.

14. Now inspect server-side variables in real time. Expand **previous** and notice the previous category value.



15. Collapse **previous.** Inspect **variable1** and **variable2** under local variables. Note they are undefined.

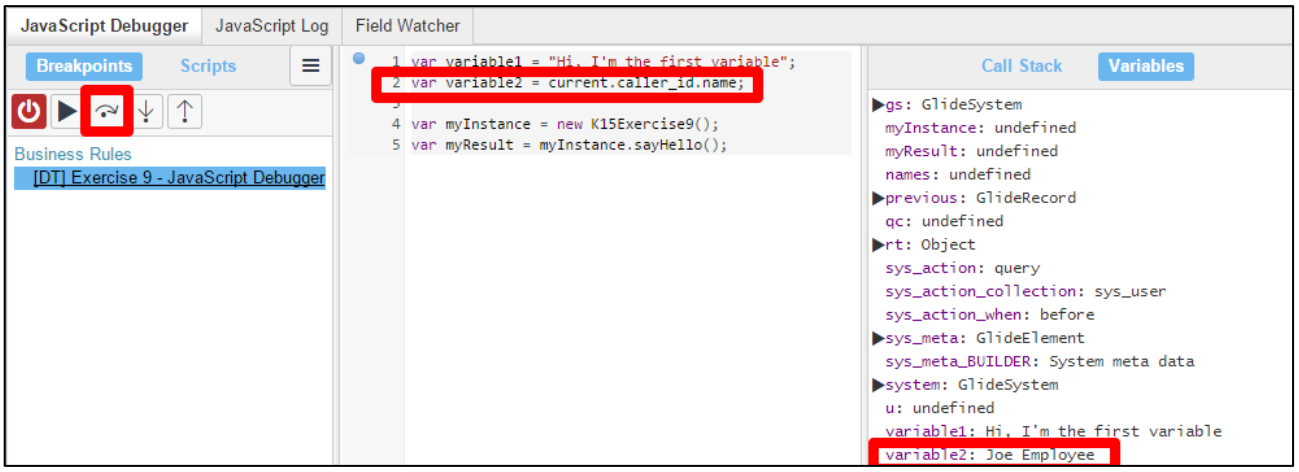
```
variable1: undefined
variable2: undefined
```

©2015 ServiceNow, Inc. All rights reserved.

16. In the left pane, click **Step over**. This moves us to the next line of code. Notice **variable1** in the Variable Inspector now has a value.

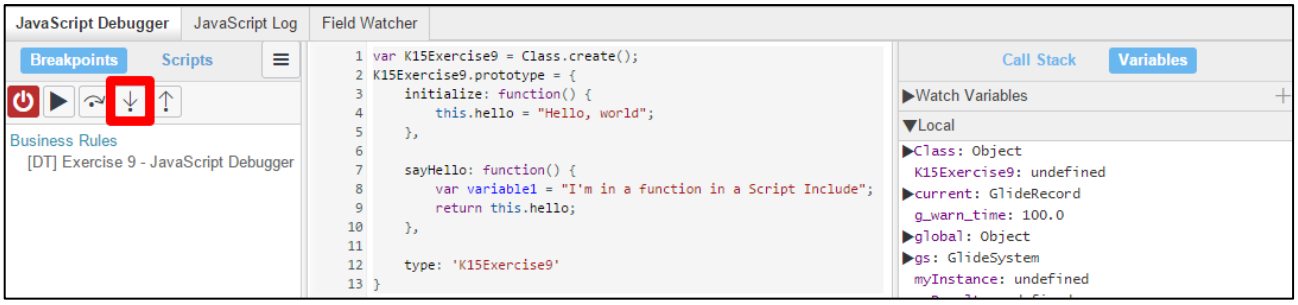


17. Click **Step over** again, and notice that **variable2** now has a value.

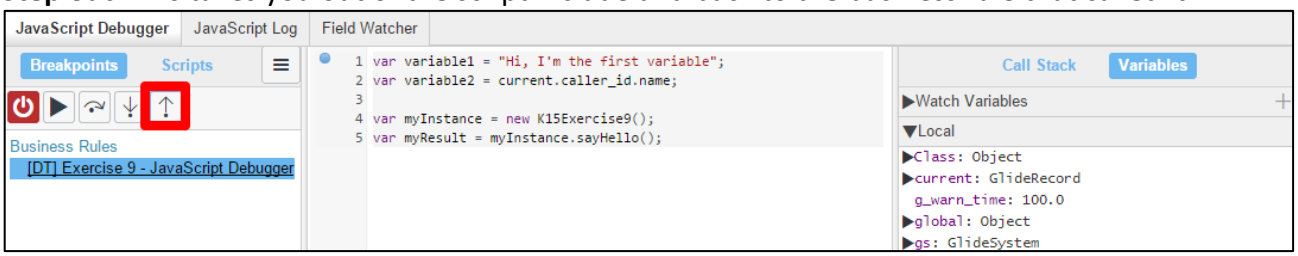


©2015 ServiceNow, Inc. All rights reserved.

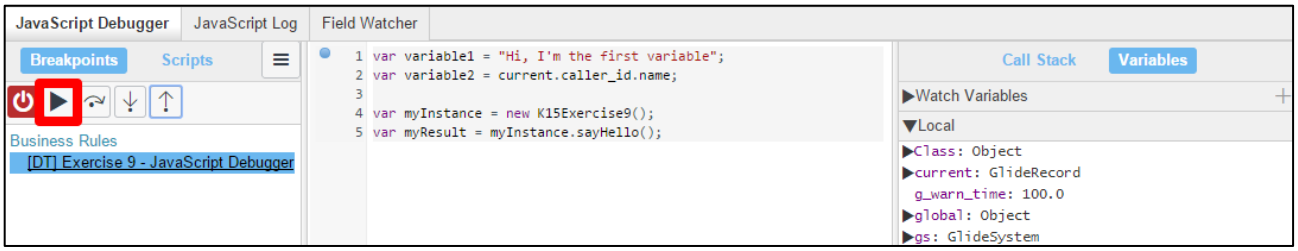
18. Next, step into the K15Exercise9 Script include. Click **Step in**. You are now inside of the K15Exercise9 Script Include. Notice that the contents of the variable inspector updated to reflect the current context.



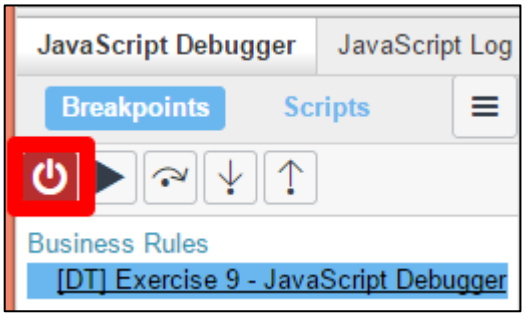
19. You could continue using stepping over each line of code in the Script Include, but instead click **Step out**. This takes you out of the script include and back to the business rule that called it.



20. Finally, click **Go**. This tells the debugger to execute all code until another breakpoint is hit. Since you have only one breakpoint and have already passed it, this means the record finishes saving and the transaction completes.



21. **Disable** the debugger



©2015 ServiceNow, Inc. All rights reserved.

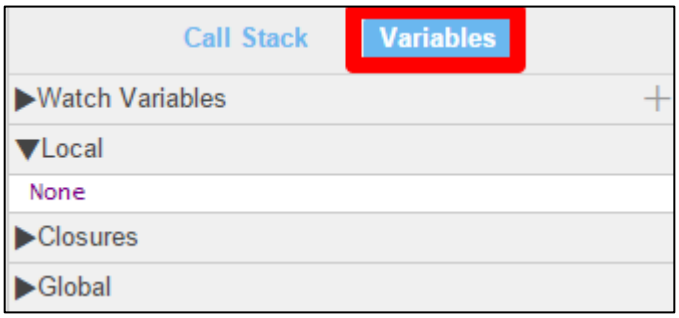
Lab Goal

This lab explains how to **Watch** variables using the Variables Inspector.

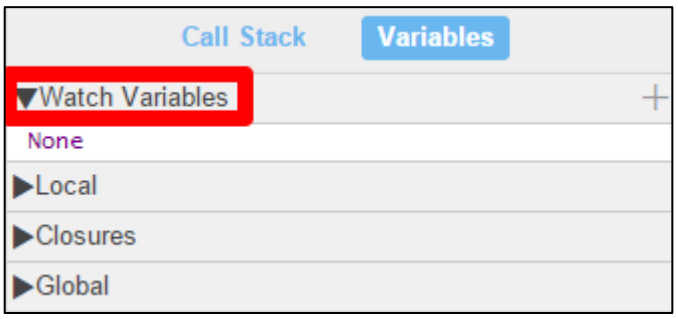
Lab 10
Watch Variables

Add Variables to Watch

1. Click **Variables**.

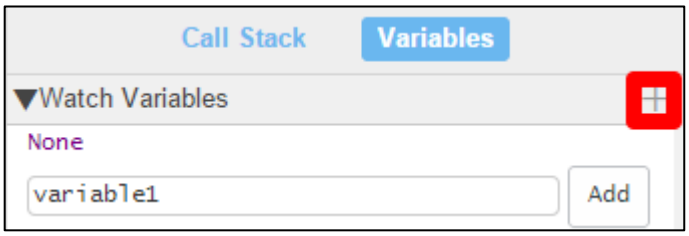


2. Expand **Watch Variables**.

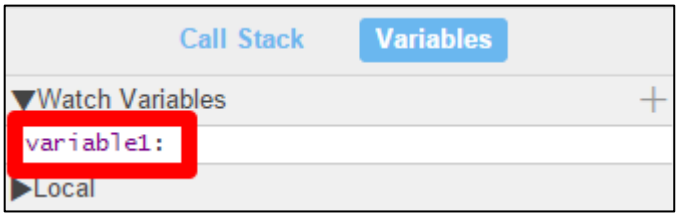


©2015 ServiceNow, Inc. All rights reserved.

3. Click the **plus (+)** sign on the right. Type **variable1** in the text box. Click **Add**.

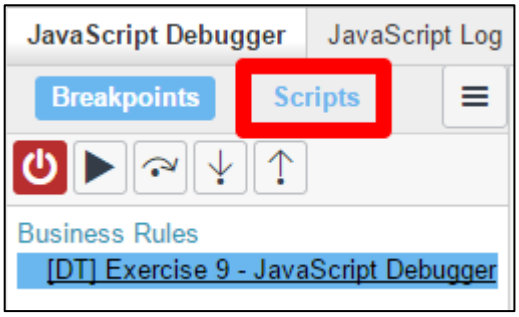


Result

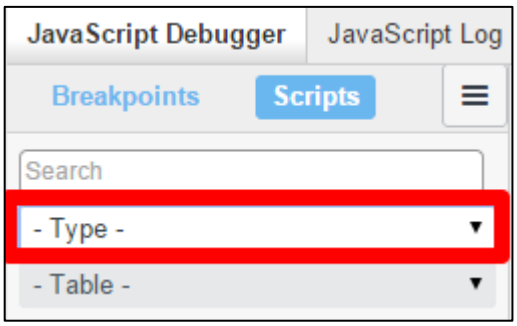


4. Repeat step 3 to **watch** a second variable called **variable2**.

5. Click **Scripts**.

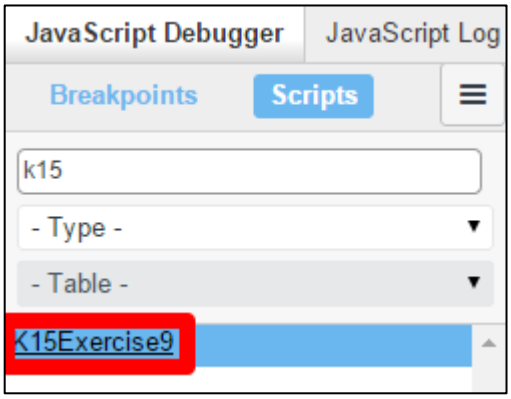


6. Clear out **Type** if there is something already selected.

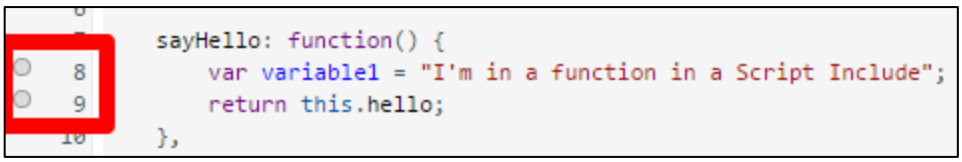


©2015 ServiceNow, Inc. All rights reserved.

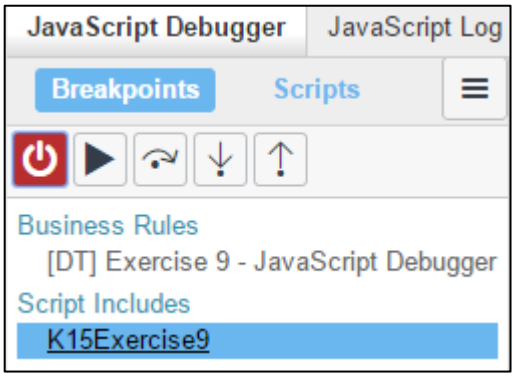
7. Type **K15** in the search box and click **K15Exercise9** from the list of results.



8. Set breakpoints on **line 8** and **line 9**.



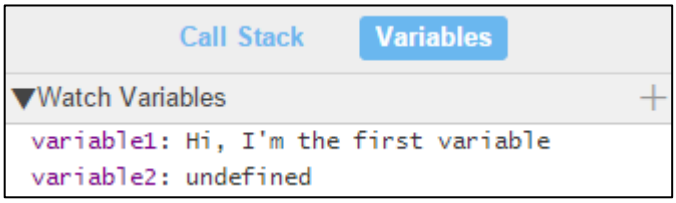
9. Switch back to the **Breakpoints** tab and **Enable** debugging. Notice that the Script Include which set breakpoints is now in the list.



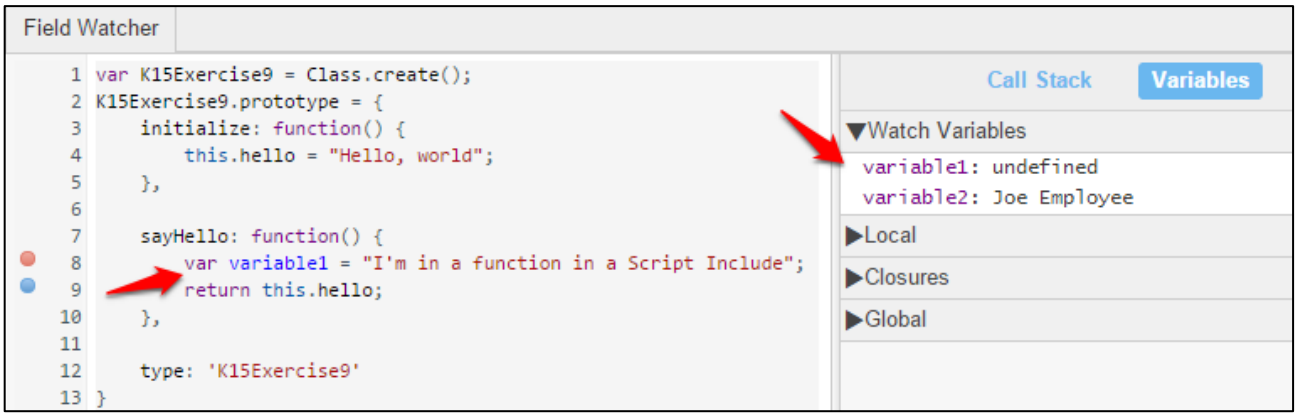
10. Modify the incident record in some way and **save** it. You return to the first breakpoint.

©2015 ServiceNow, Inc. All rights reserved.

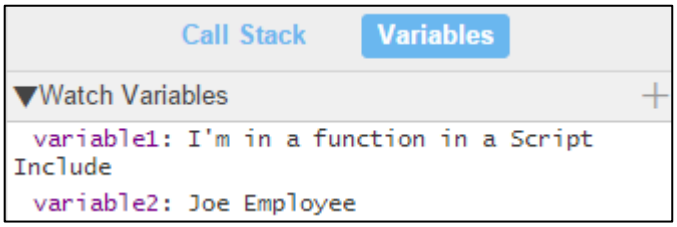
11. Click **Step over.** The first watched variable now has a value.



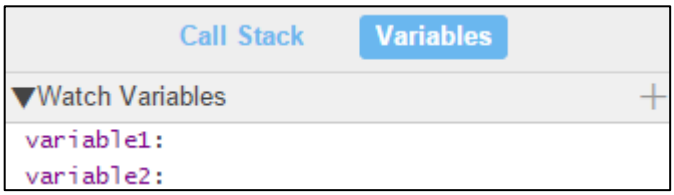
12. Click **Go**. The debugger jumps into the K15Exercise9 Script Include. Click **Go** again. Notice how the watched variables change.



13. Click **Go** again.



14. Click **Go** a final time. The transaction ends, and the watched variables are empty again.



Summary: Use the JavaScript debugger to interactively inspect the contents of your variables and the execution state of the ServiceNow environment. Avoid unnecessary logging by going straight to the debugger and stepping through the Business Rules and Script Includes.

©2015 ServiceNow, Inc. All rights reserved.