knowledge14
April 27–May 1, 2014 • Moscone Center • San Francisco, CA

Lab Guide

Debugging 201:

Practical Troubleshooting Concepts and Tools

Chris Maloy & Josh Nerius

Lab instance: https://**<<name>>**.lab.service-now.com

Login / Passwords:

admin / Knowledge14

itil / Knowledge14

employee / Knowledge14

Lab Goal

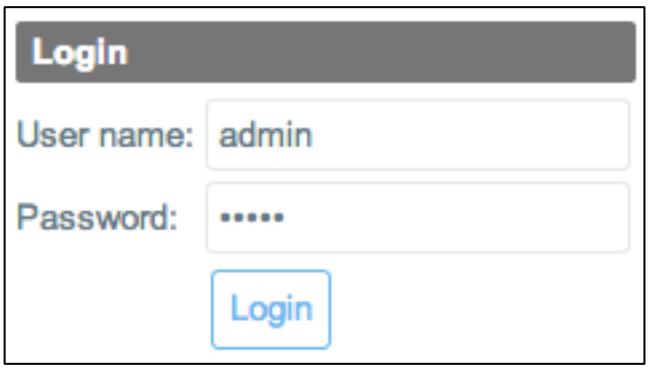
ServiceNow provides a variety of Debug options. This lab will explain how to enable and view output from the **Debug Business Rule** debug module.

Lab 1
Debugging Business Rules

Log on to Your Lab Instance

1. Your instance URL is on the cover page.

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



®2014 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 on 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, we will get to this soon).



3. Repeat the steps from "**Open an Existing Incident**".

©2014 ServiceNow, Inc. All rights reserved.

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")

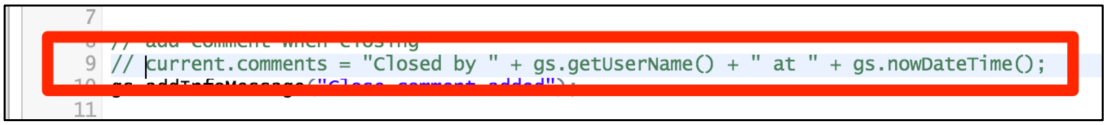
5. From the debug output, **click on the business rule name** to open it.



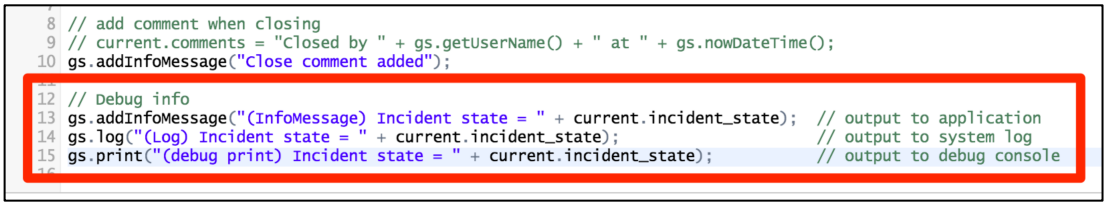
Remove the Business Rule Condition

1. From the Business Rule form, **Remove** the condition so the rule will execute.

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



3. **Uncomment** the existing debugging statements as shown.



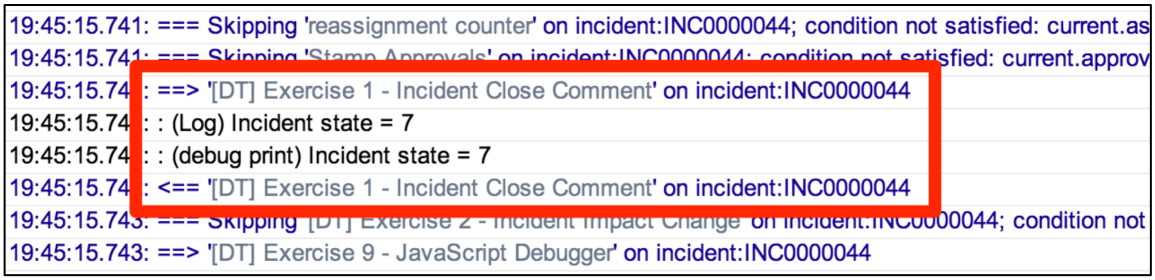
4. **Navigate** to another open incident.

5. Set the **State** to **Closed**, set **Close code** and **Close notes.**

©2014 ServiceNow, Inc. All rights reserved.

6. **Save** the incident.

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



8. It looks like we used the wrong value in our Business Rule condition!

©2014 ServiceNow, Inc. All rights reserved.

Fixing the Business Rule

1. First, correct the condition.

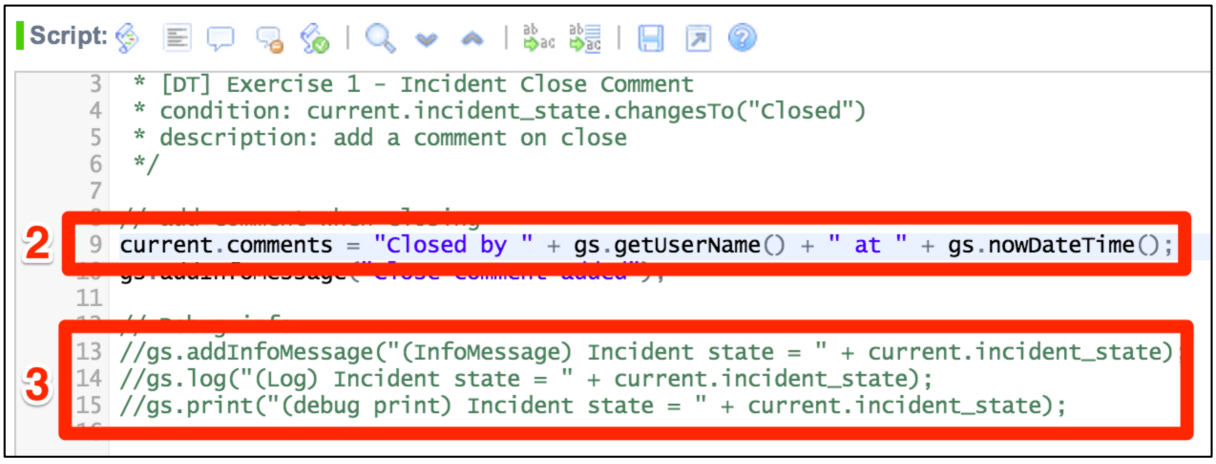
   ```
   current.incident_state.changesTo('7')
   ```



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

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

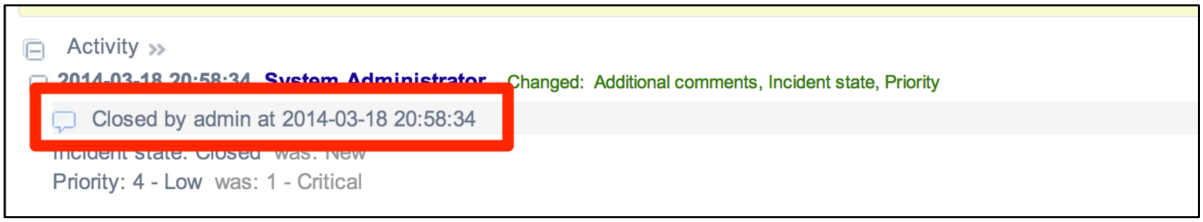
   Comment out lines 13, 14 and 15 from the existing debugging statements as shown.



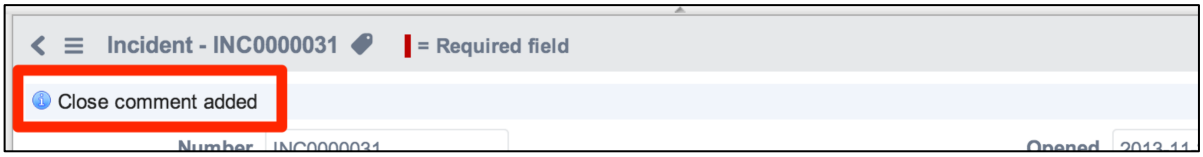
©2014 ServiceNow, Inc. All rights reserved.

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.



©2014 ServiceNow, Inc. All rights reserved.

Lab Goal

Utilize **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.

Lab 2 Debugging Business Rules (Details)

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.



©2014 ServiceNow, Inc. All rights reserved.

Lab Goal

Utilize try/catch in JavaScript to handle errors.

Utilize 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.

Lab 3
Debug Log
Try / Catch

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.

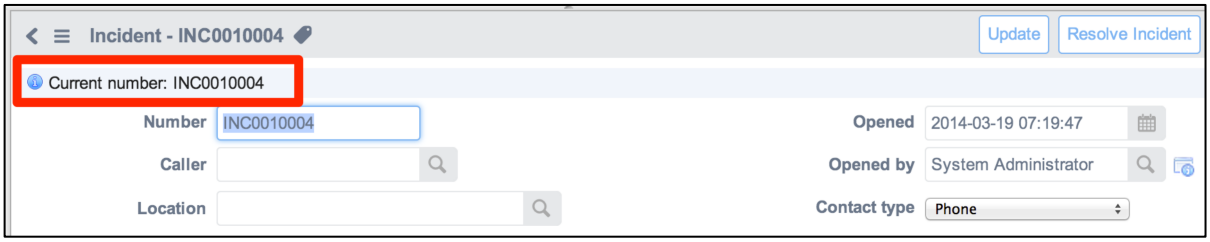
```
9    //output current number
10   gs.addInfoMessage("Current number: " + current.number);
11
12   //output current short_description
13   gs.addInfoMessage("Current Short Description: " + currrent.short_description);
14
15   //output current category
16   gs.addInfoMessage("Current Category: " +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

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

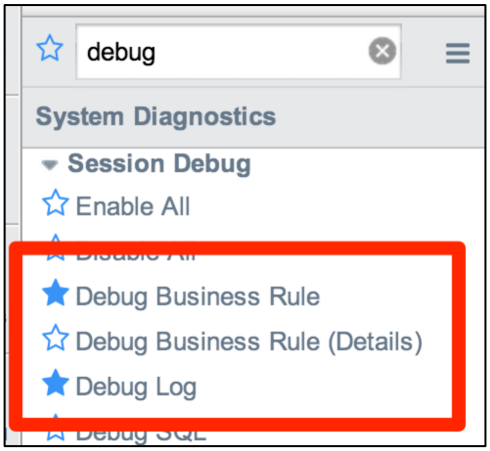
3. Add a **short description** and save the incident.

©2014 ServiceNow, Inc. All rights reserved.

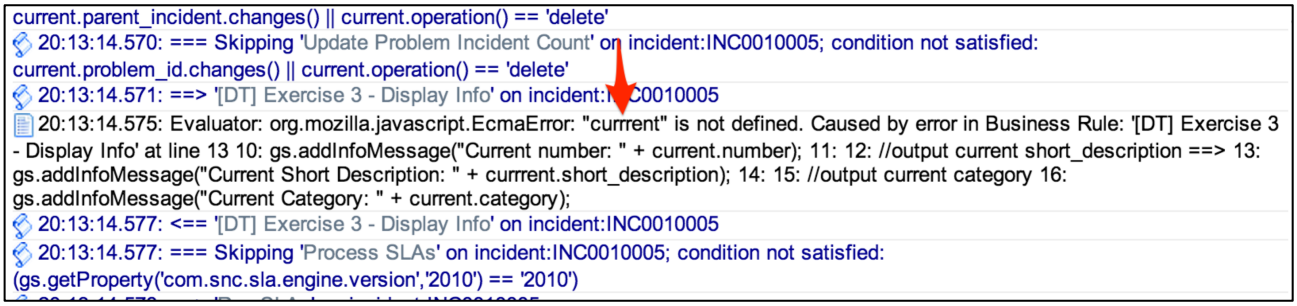
4. **Result:** Notice we are getting one message at the top of the form but not the other two.



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



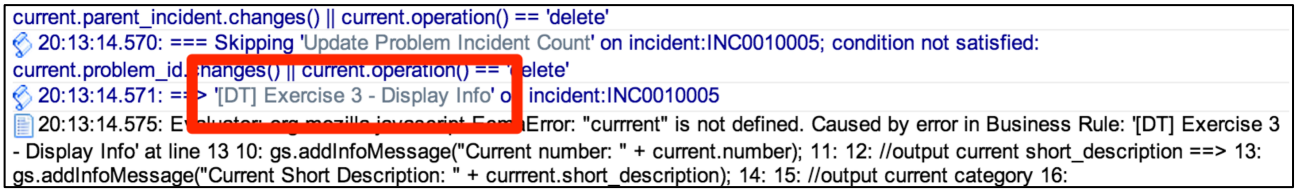
6. **Create another software incident** and view the debug output. Notice there is a script exception because we have an extra “r” in “current” (do not fix this yet).



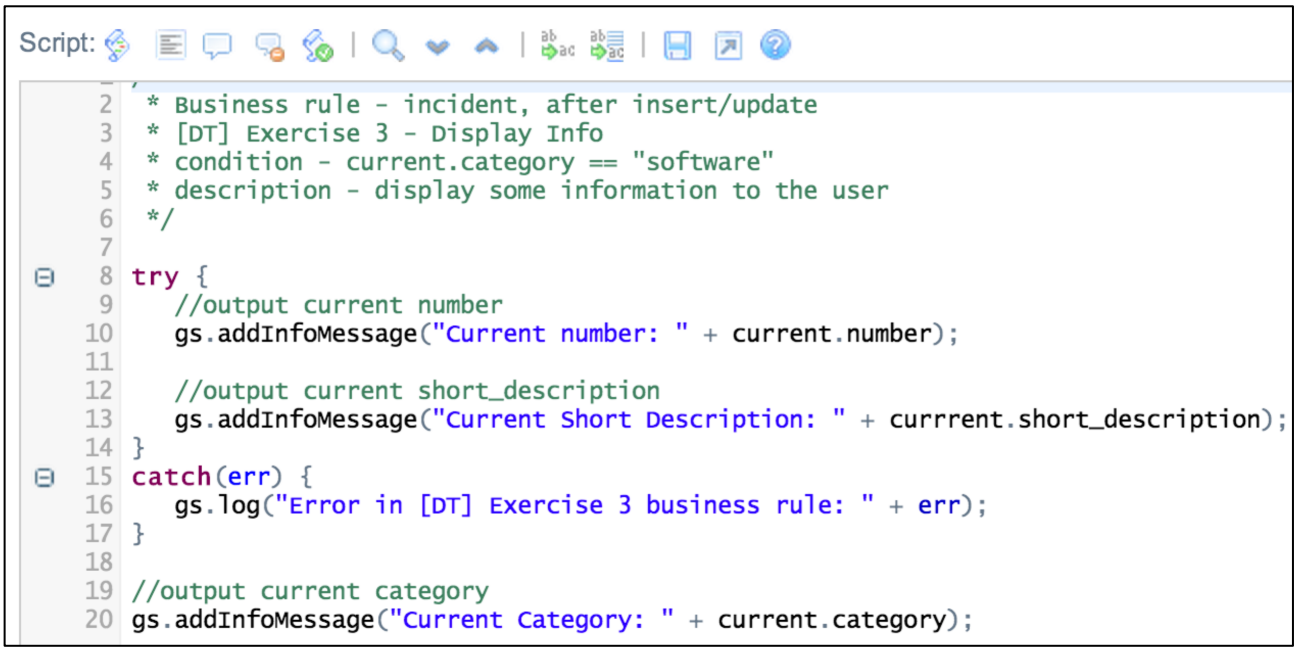
©2014 ServiceNow, Inc. All rights reserved.

Adding a try … catch block

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

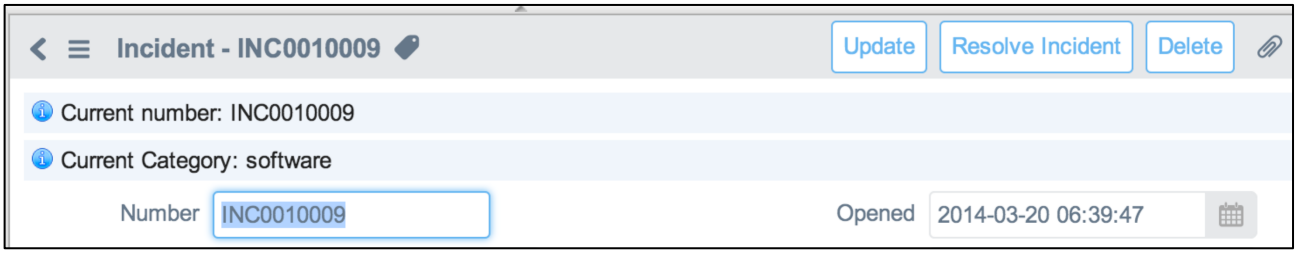


2. **Add a try catch block** around our statements and save the business rule. Notice that we have left one statement below the catch block.

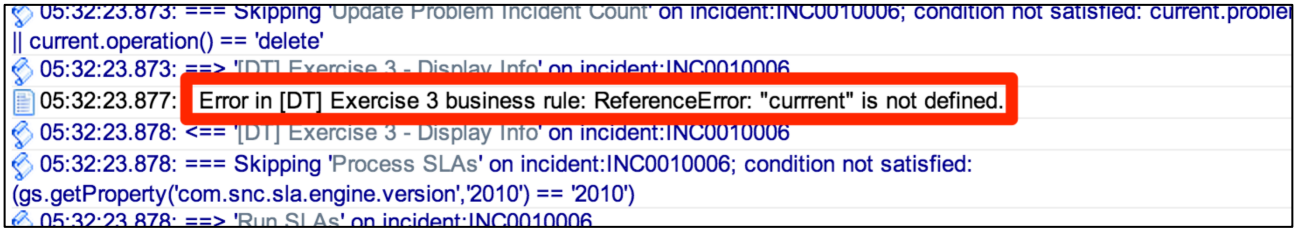


©2014 ServiceNow, Inc. All rights reserved.

3. Create one more software incident and save it. We are now getting 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, giving us our second info message.



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.

©2014 ServiceNow, Inc. All rights reserved.

Lab Goal

Use the JavaScript debugger console.

Lab 4 Client Script Debugging

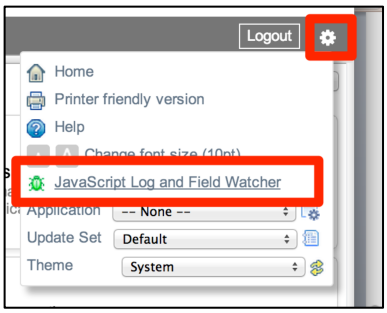
Enabling the JavaScript Debugger Console

1. The Incident table has an onChange 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**".



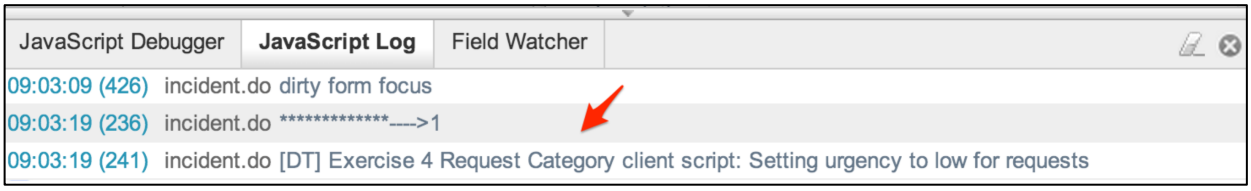
©2014 ServiceNow, Inc. All rights reserved.

3. **Create a new Incident**, set **Urgency** to **2 – Medium.**

4. Change the **Category** to **Request.**

5. This causes the onChange script to fire, and you will see the jslog statement in the Debugger output.

6. Review debug console.



©2014 ServiceNow, Inc. All rights reserved.

Lab Goal

Utilize 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!



©2014 ServiceNow, Inc. All rights reserved.

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.

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

   Change code starting at line 18**.**

   Before:

```
15        callerField.setStyle({color: ""});
16        return;
17    }
18    var caller = g_form.getReference('caller_id');
19
20    var callerField = $('sys_display.incident.caller_id');
21    if (!callerLabel || !callerField)
22        return;
23
```

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

```
Script:
16        return;
17    }
18    var caller = g_form.getReference('caller_id', function(caller) {
19        // Your code here
20        var callerLabel = $('label.incident.caller_id');
21        var callerField = $('sys_display.incident.caller_id');
22        if (!callerLabel || !callerField)
23            return;
24
25        //check for VIP status
26        if (caller.vip == 'true') {
27            callerLabel.setStyle({backgroundImage: "url(images/icons/vip.gif)", bac
28            callerField.setStyle({color: "red"});
29        } else {
30            callerLabel.setStyle({backgroundImage: ""});
31            callerField.setStyle({color: ""});
32        }
33    });
34 }
```

©2014 ServiceNow, Inc. All rights reserved.

2. What did we do? We turned the blocking call into a non-blocking call with a callback. In this example, we used an **anonymous** function.

3. Another way we could have achieved the same result is to create a **named** function. This can be achieved by using code similar to the following example:

```
19    var caller = g_form.getReference('caller_id', vipCallback);
20
21    function vipCallback(caller) {
22        // Your code here
23    }
24
```

©2014 ServiceNow, Inc. All rights reserved.

Lab Goal

Utilize 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.

Script:

```
/*
 * 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);
```

©2014 ServiceNow, Inc. All rights reserved.

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

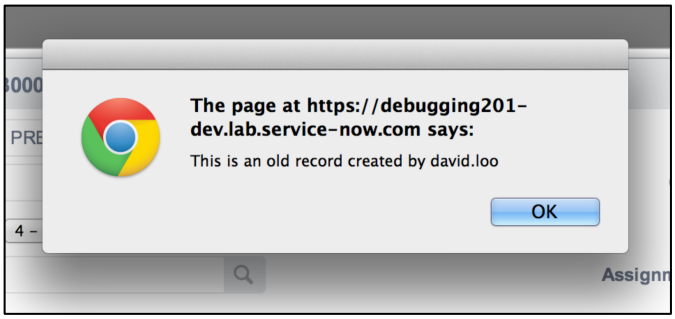
Script:

```
1  /*
2   * Client Script - problem, onLoad
3   * [DT] Exercise 6 - Notify on old problems
4   * Description - get data from display business rule and alert user on old problems
5   */
6
7  function onLoad() {
8      // get data from display rule scratchpad
9      var createdBy = g_scratchpad.created_by;
10     var daysOld = g_scratchpad.days_old;
11
12     if (parseInt(daysOld) > 30) {
13         alert("This is an old record created by " + createdBy);
14     }
15 }
```

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.

©2014 ServiceNow, Inc. All rights reserved.

Lab Goal

Use setLimit() to reduce the number of records returned by a GlideRecord query.

Lab 7
GlideRecord setLimit()

Without setLimit()

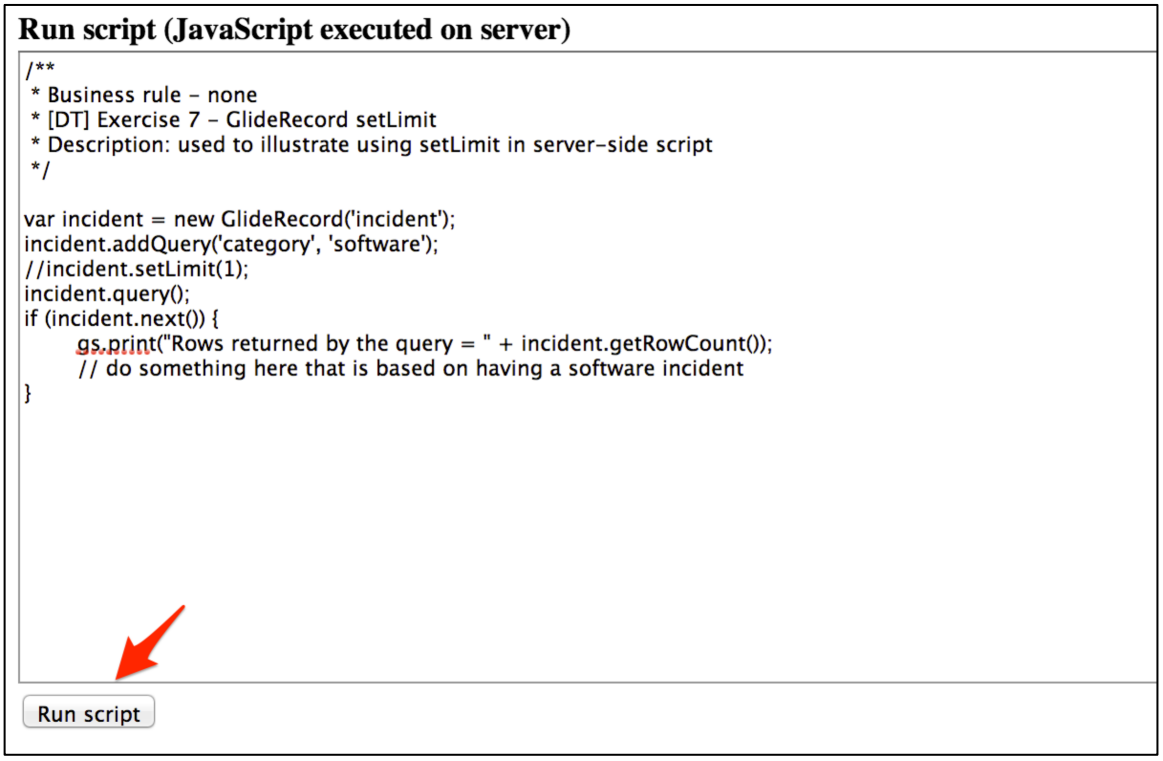
1. Navigate to **System Definition > Business Rules**.
2. Open **[DT] Exercise 7 - GlideRecord setLimit**.
3. **Copy** the contents of the Script field.
4. Activate the "**security_admin**" elevated privilege.



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

©2014 ServiceNow, Inc. All rights reserved.

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



7. Notice the output (this number will vary depending on how many software incidents are in your instance).

```
*** Script: Rows returned by the query = 16
```

©2014 ServiceNow, Inc. All rights reserved.

A better way

Our previous query will get slower and slower as the number of records in the system increases. To mitigate this, we use setLimit().

1. Go back to **Scripts – Background**.

2. Paste the script again/

3. Uncomment this line:

   setLimit(1);

4. Run again.

5. Notice only one record is returned this time. Much better!

```
[0:00:00.017] Script completed: script
*** Script: Rows returned by the query = 1
```

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.

©2014 ServiceNow, Inc. All rights reserved.

Lab Goal

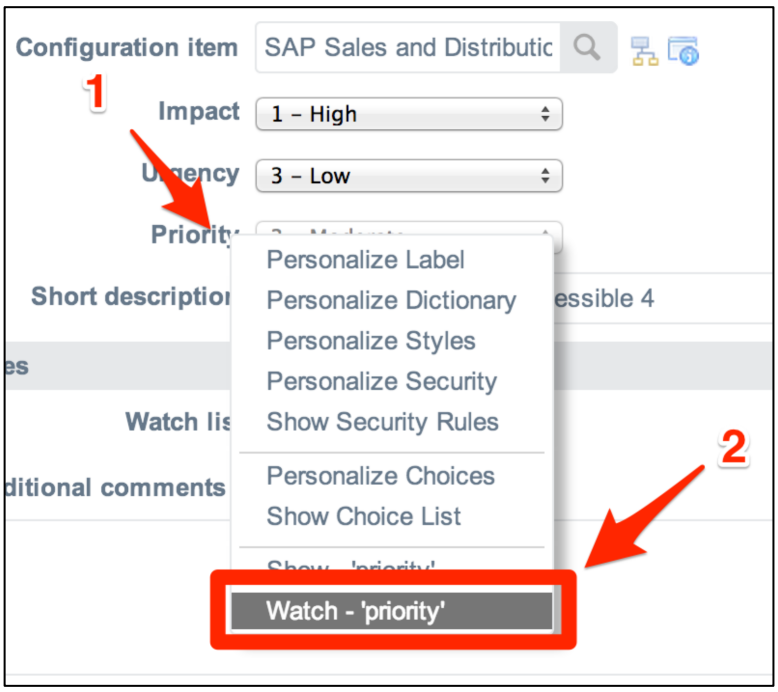
Use the Field Watcher to determine why a field value is changing.

Lab 8
Field Watcher

Enable Field Watcher

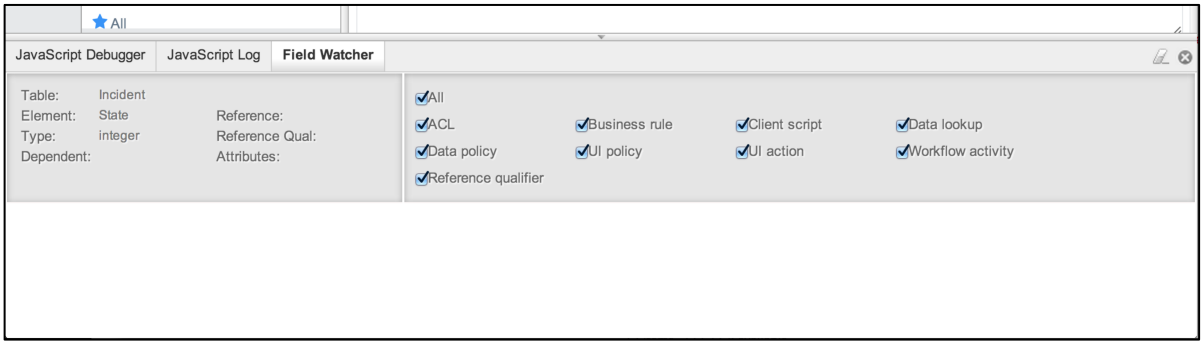
1. Open an existing Incident record.

2. Right-click the **Priority** label, click **"Watch – 'priority'".**

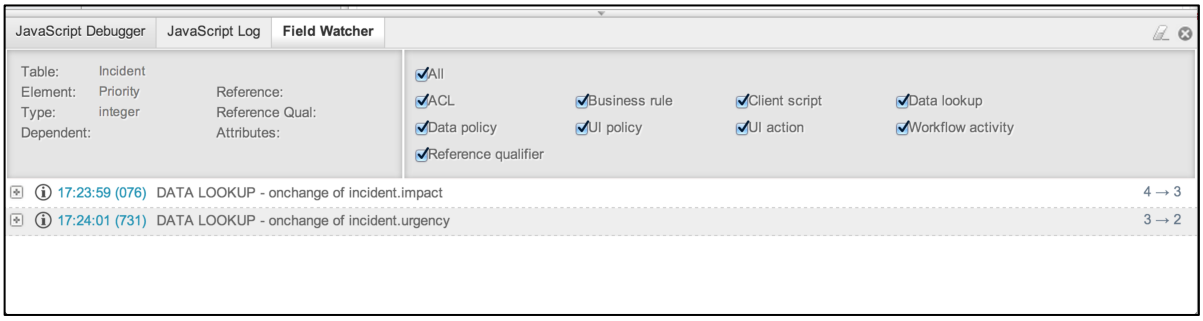


©2014 ServiceNow, Inc. All rights reserved.

3. The field watcher will appear at the bottom of the screen.



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

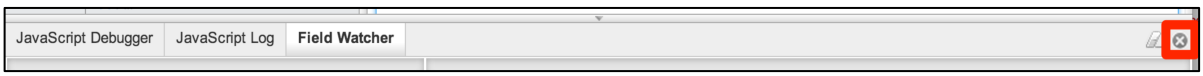


5. We see that a data lookup rule is setting the Impact whenever Impact or Urgency change.

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.



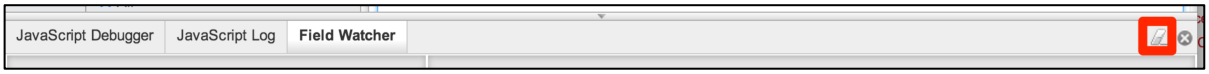
©2014 ServiceNow, Inc. All rights reserved.

Optional – Explore Some More

1. Create a new incident record, set the **Category** to **Software**, **Impact** to **1**, **Urgency** to **1**, **Save**.

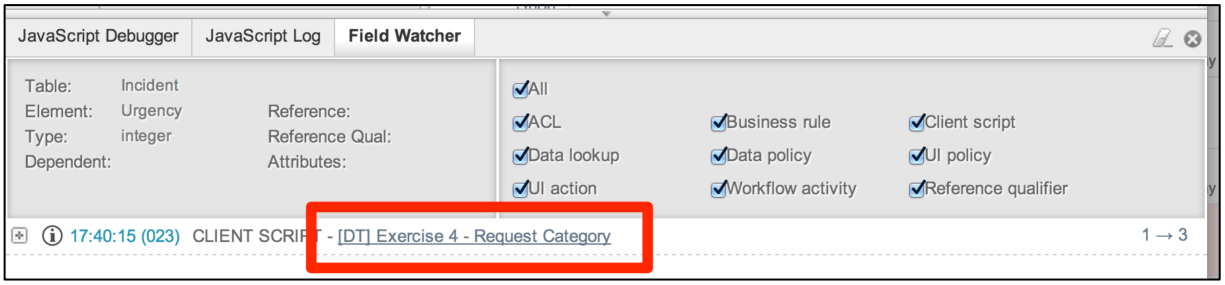
2. Enable the Field Watcher for the **Urgency** field.

3. Clear the field watcher output by clicking the **eraser** icon.



4. Change the **Category** to **Request.**

5. We see that our earlier client script is to blame for the urgency change. If desired, we can click on the script name to go directly to the Client Script.



©2014 ServiceNow, Inc. All rights reserved.

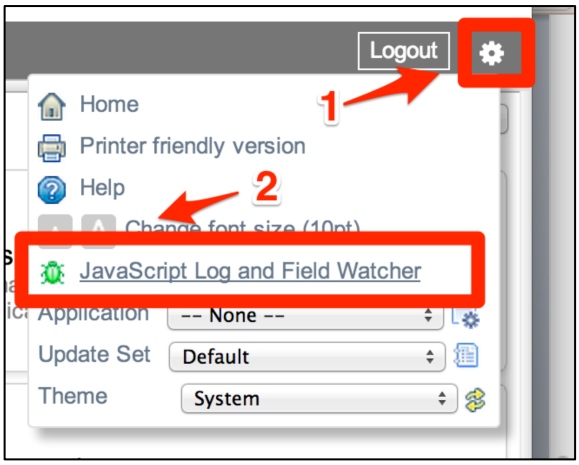
Lab Goal

Learn to use the JavaScript debugger

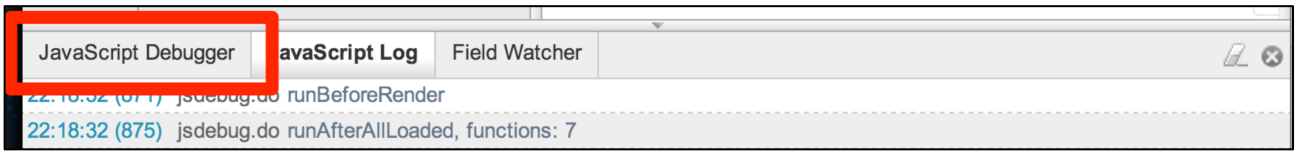
Lab 9
JavaScript Debugger

Setting a breakpoint

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

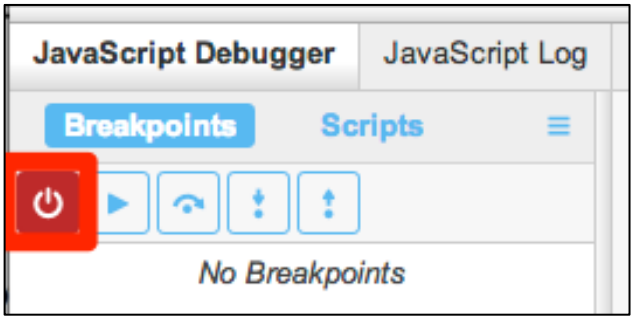


2. If necessary, change to the **JavaScript Debugger** tab.

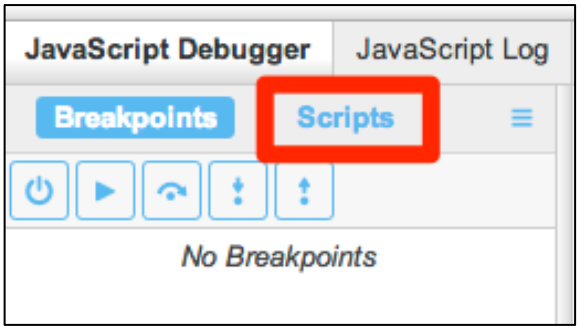


©2014 ServiceNow, Inc. All rights reserved.

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

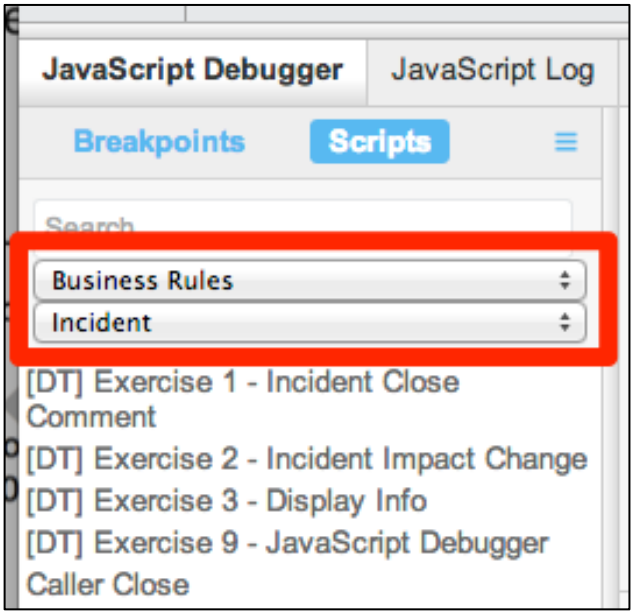


4. Click **Scripts.**

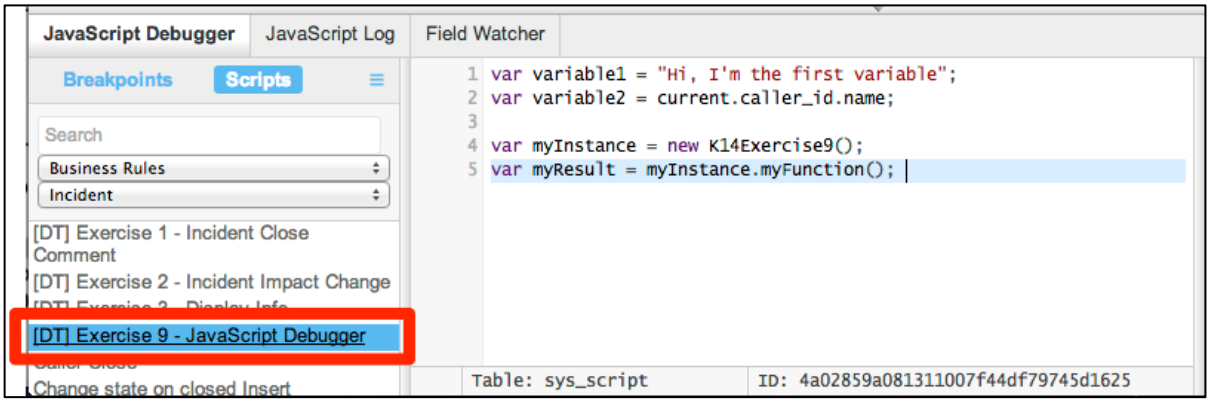


®2014 ServiceNow, Inc. All rights reserved.

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

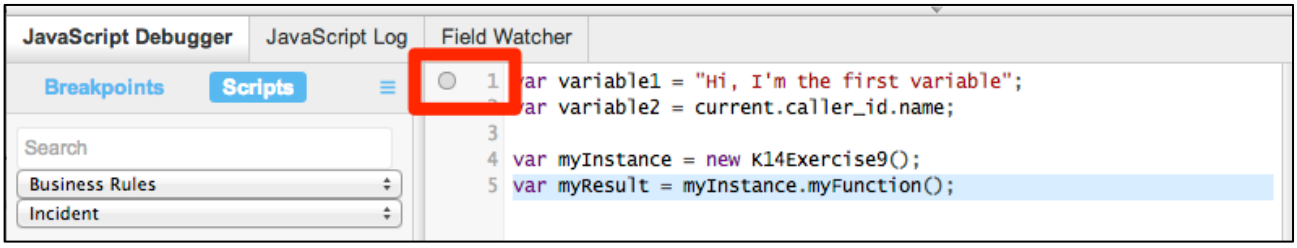


6. Click **[DT] Exercise 9 – JavaScript Debugger** from the list of scripts.



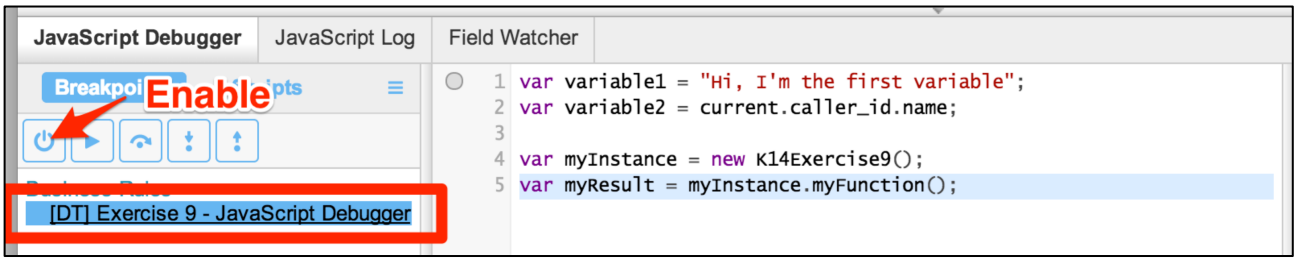
©2014 ServiceNow, Inc. All rights reserved.

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.

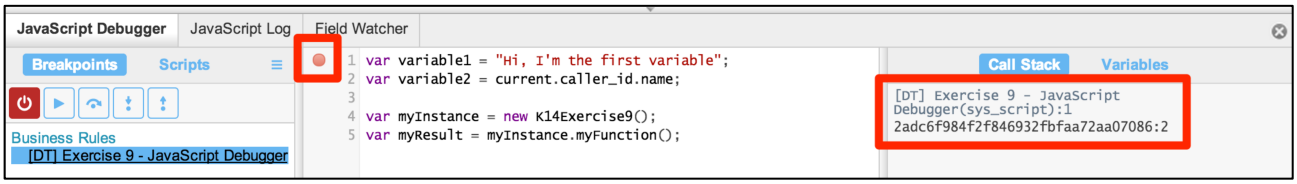
9. Change back to the **Breakpoints** subtab and **Enable** debugging. Notice the business rule now in the list. Any time we set a breakpoint, the related record will show up in the **Breakpoints** tab.



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

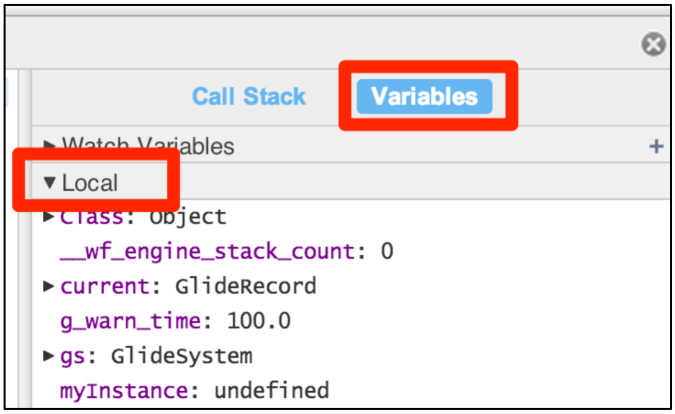
11. Right click the header and **Save.**

12. The breakpoint indicator should turn red, and the page will remain in a loading state. The **Call Stack** indicates where we have stopped.

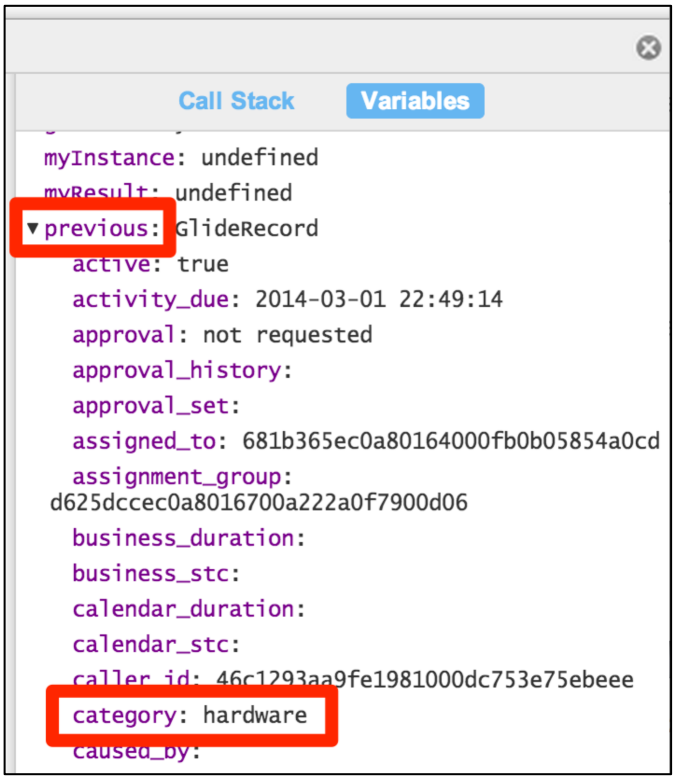


©2014 ServiceNow, Inc. All rights reserved.

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



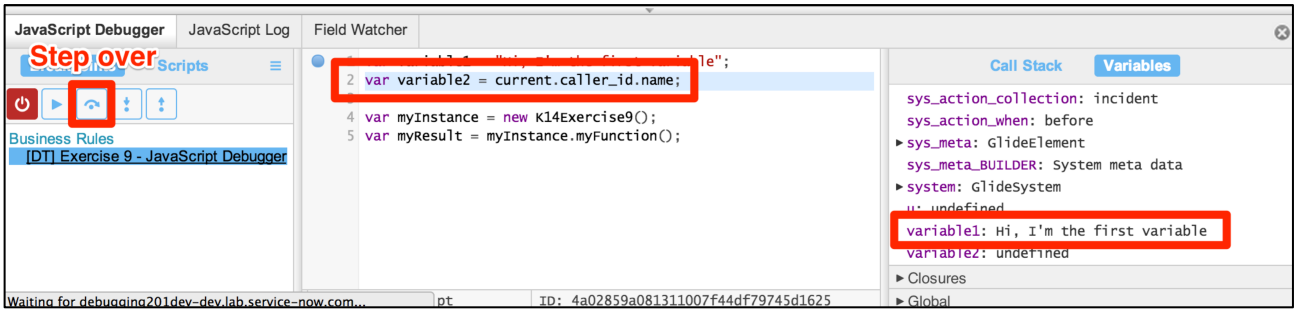
14. We can now inspect server-side variables in real time. Expand **previous** and notice the previous category value.



©2014 ServiceNow, Inc. All rights reserved.

15. Collapse **previous.**

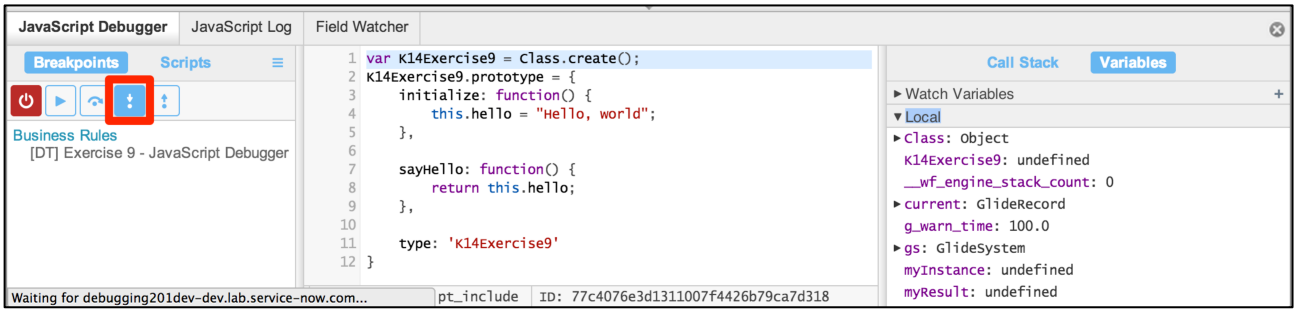
16. In the left pane, click **Step over**. This moves us to the next line of code. Notice **variable1** in the Variable Inspector now has a value.



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

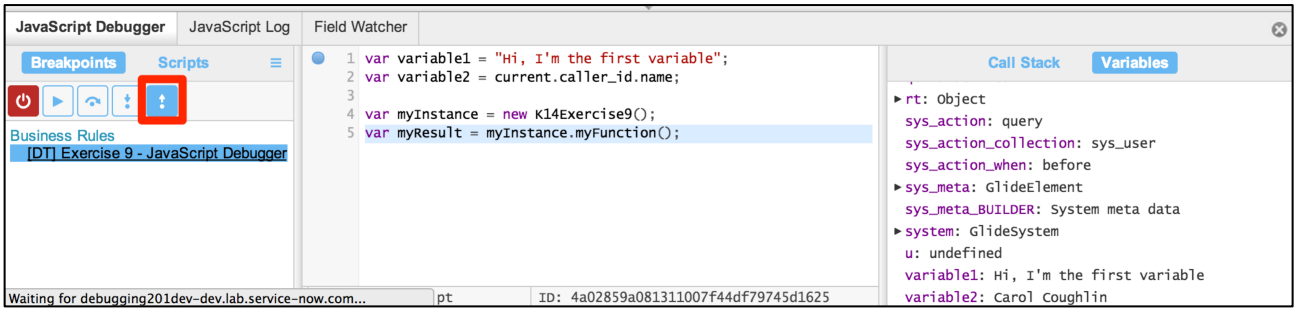


18. Now, we want to step **into** the K14Exercise9 Script include. Click **Step in**. We are now inside of the K14Exercise9 Script Include. Notice that the contents of the variable inspector have updated to reflect our current context.

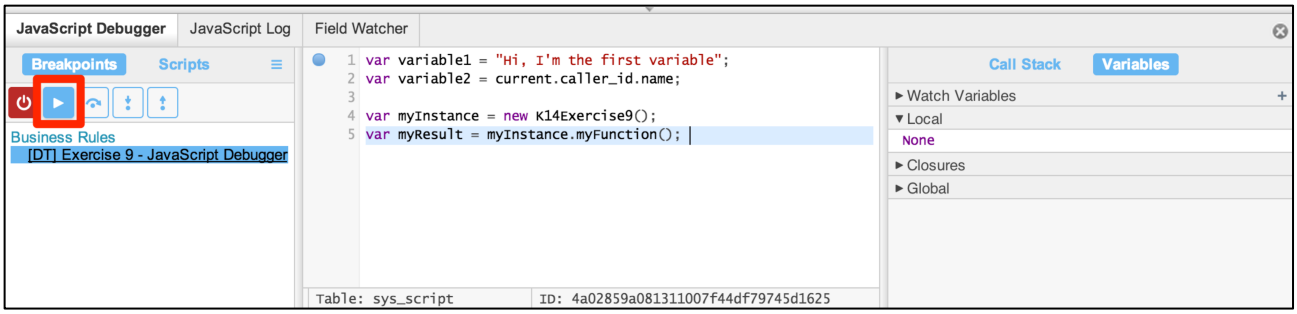


©2014 ServiceNow, Inc. All rights reserved.

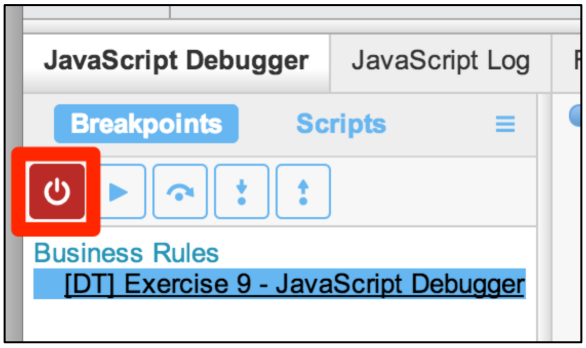
19. We could continue using stepping over each line of code in the Script Include, but instead we will click **Step out**. This takes us out of the script include and back to the business rule that called it.



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



21. **Disable** the debugger



©2014 ServiceNow, Inc. All rights reserved.

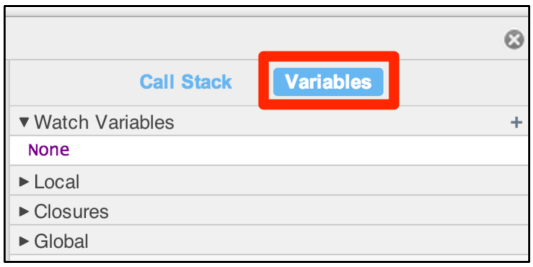
Lab Goal

Watch variables using the Variables Inspector.

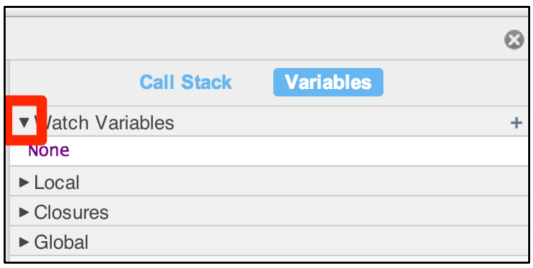
Lab 9.1
Watching Variables

Add Variables to Watch

1. Click **Variables.**

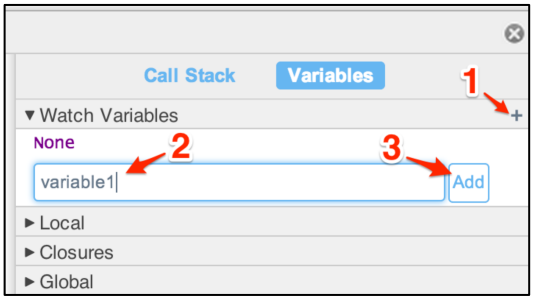


2. Expand **Watch Variables.**

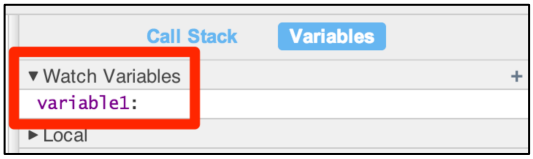


©2014 ServiceNow, Inc. All rights reserved.

3. Click the **plus** sign on the right, type **variable1** and click **Add**.

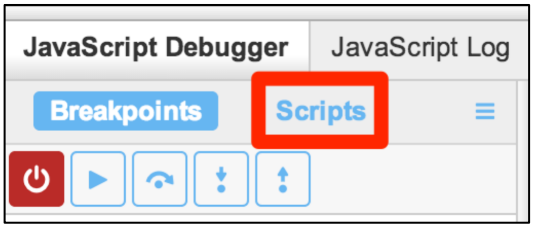


Result



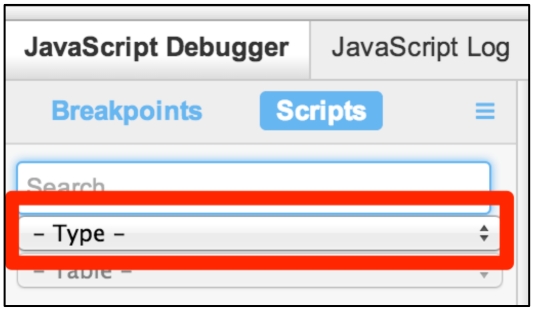
4. Repeat step 3 to watch a second variable called **variable2**.

5. Click **Scripts**.

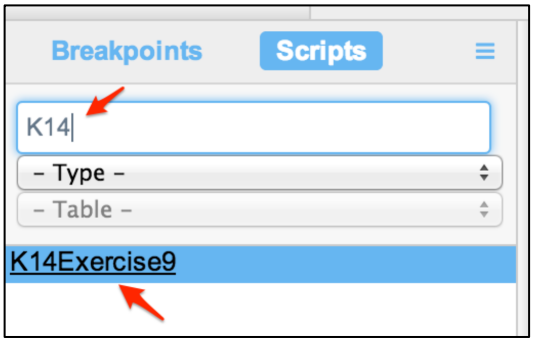


©2014 ServiceNow, Inc. All rights reserved.

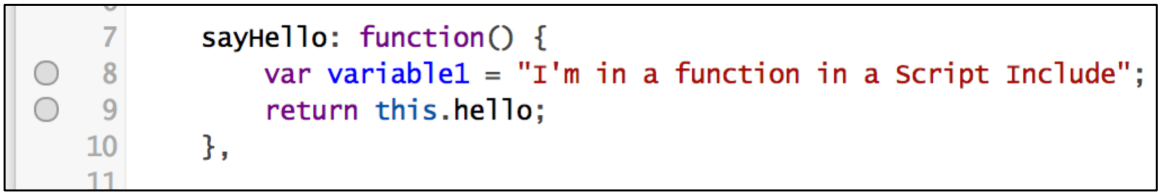
6. Clear out **Type** if there is something already selected.



7. Type **K14** in the search box and click K14Exercise9 from the list of results.

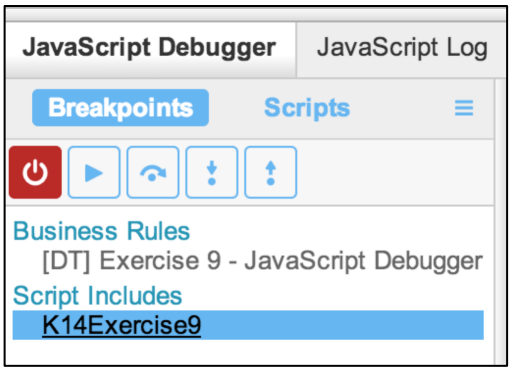


8. Set breakpoints on **line 8** and **line 9**.



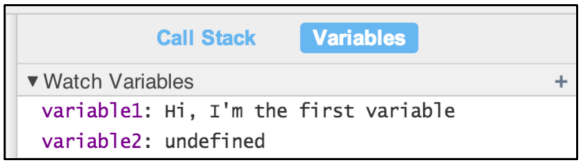
©2014 ServiceNow, Inc. All rights reserved.

9. Switch back to the **Breakpoints** tab and **Enable** debugging. Notice the script include we just set breakpoints in is now in the list.

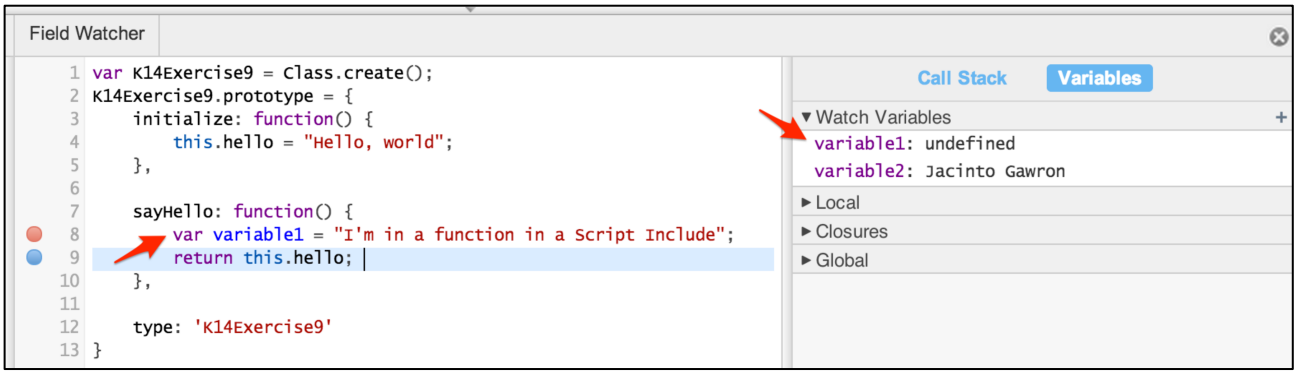


10. Modify the incident record in some way and **save** it. We will be taken to the first breakpoint.

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



12. Click **Go**. The debugger jumps into the K14Exercise9 Script Include. Click **Go** again. Notice how our watched variables change.

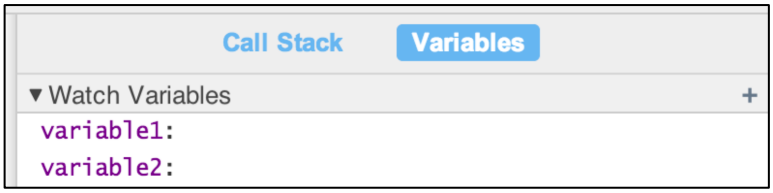


©2014 ServiceNow, Inc. All rights reserved.

13. Click **Go** again.



14. Click **Go** one 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 your Business Rules and Script Includes.

©2014 ServiceNow, Inc. All rights reserved.