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

Lab Guide

Scripting 202:

Server-side Scripting

Andrew Venables & Chris Terzian

Default Login / Password:

admin / Knowledge15

itil / Knowledge15

employee / Knowledge15

This

Page

Intentionally

Left

Blank

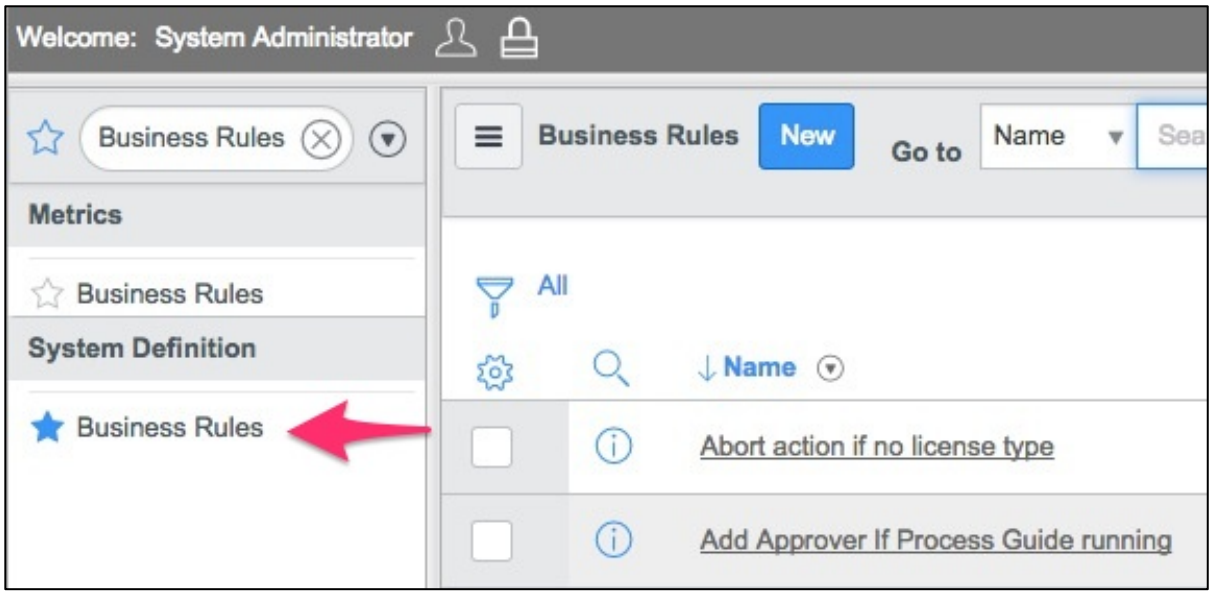
Lab Goal

This lab explores techniques for writing robust business rules and preventing pollution of the global scope.

Lab 1
Business
Rule Scope

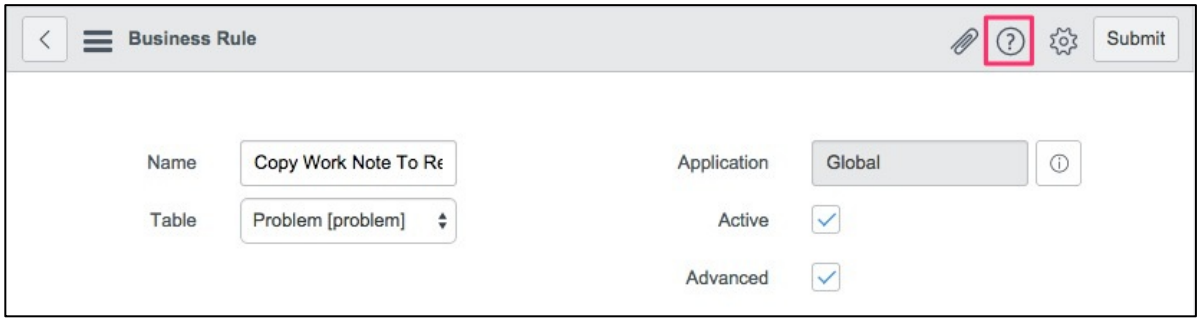
Variable Collisions

1. Navigate to **System Definition > Business Rules** and search by **Name** for **Copy Work Note To Related Incidents**.



2. Open the **Business Rule** and click the **Advanced** tab.

3. Hide form annotations if they are present by clicking on the **?** icon.

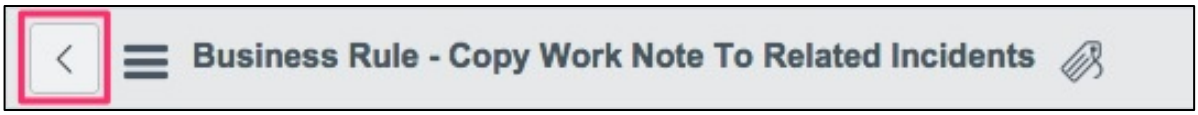


4. View the Business Rule **Script** and note that it uses a GlideRecord named **"gr"** and does an **update** operation on the incident table.

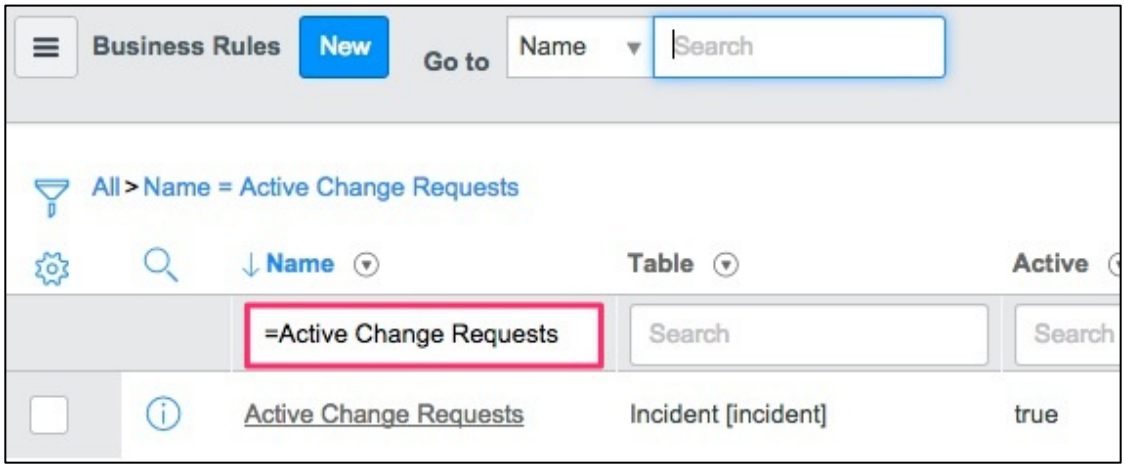
```
var gr = new GlideRecord("incident");
gr.addQuery("problem_id", current.sys_id);
gr.query();

while(gr.next()) {
   gs.addInfoMessage("Copying Work Note to " + gr.getDispla
   gr.comments = current.work_notes;
   gr.update();
}
```

This Business Rule copies work notes from the Problem record to any related/child incidents.

5. Press the **back arrow** (**<**)to return to the list of **Business Rules**.

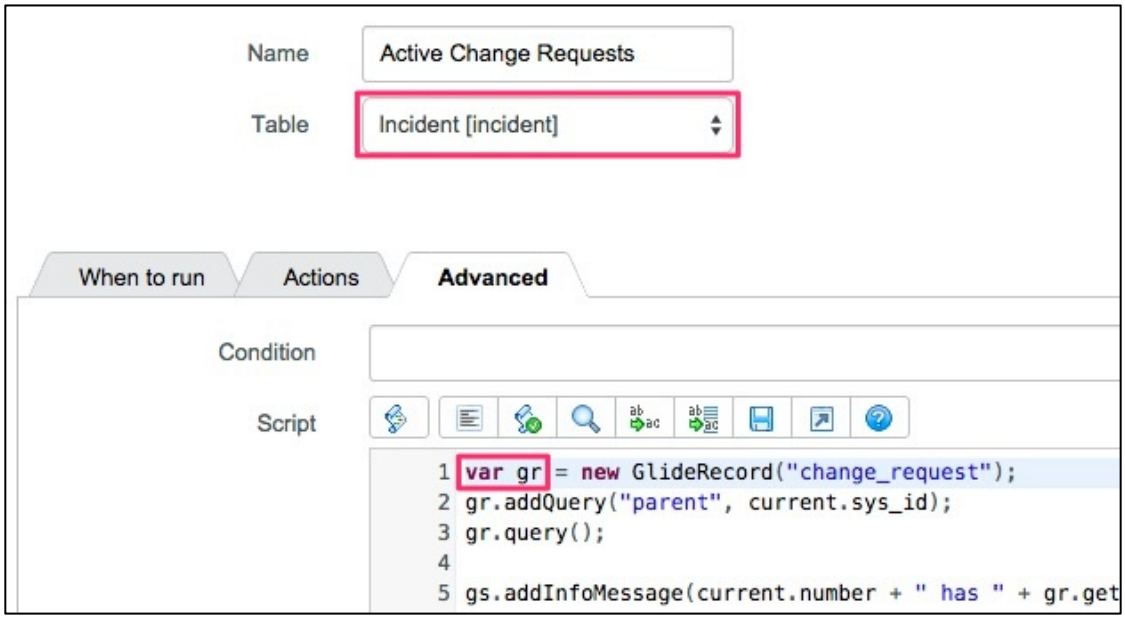


6. Search by **Name** for **Active Change Requests**.



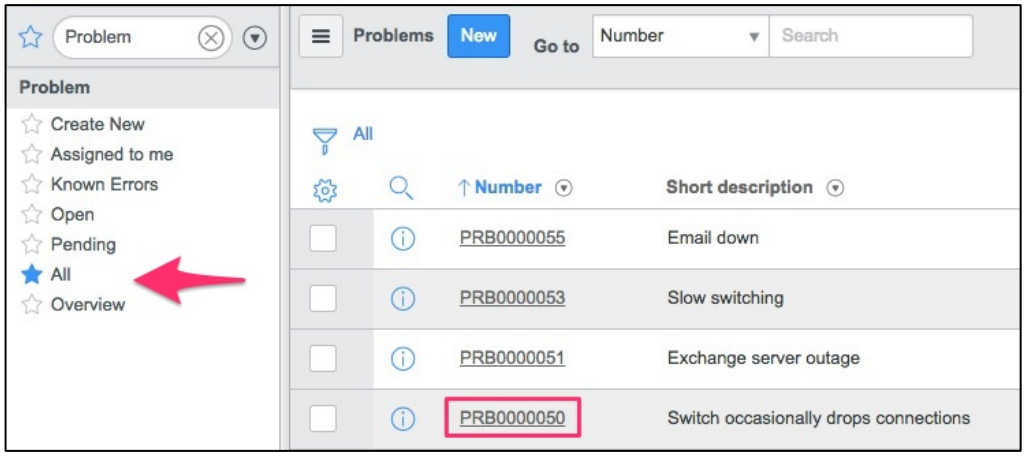
7. Open the **Business** Rule and click the **Advanced** tab.

8. Note that the Business Rule applies to the **Incident** table and that the **Script** also uses a variable **"gr"**.

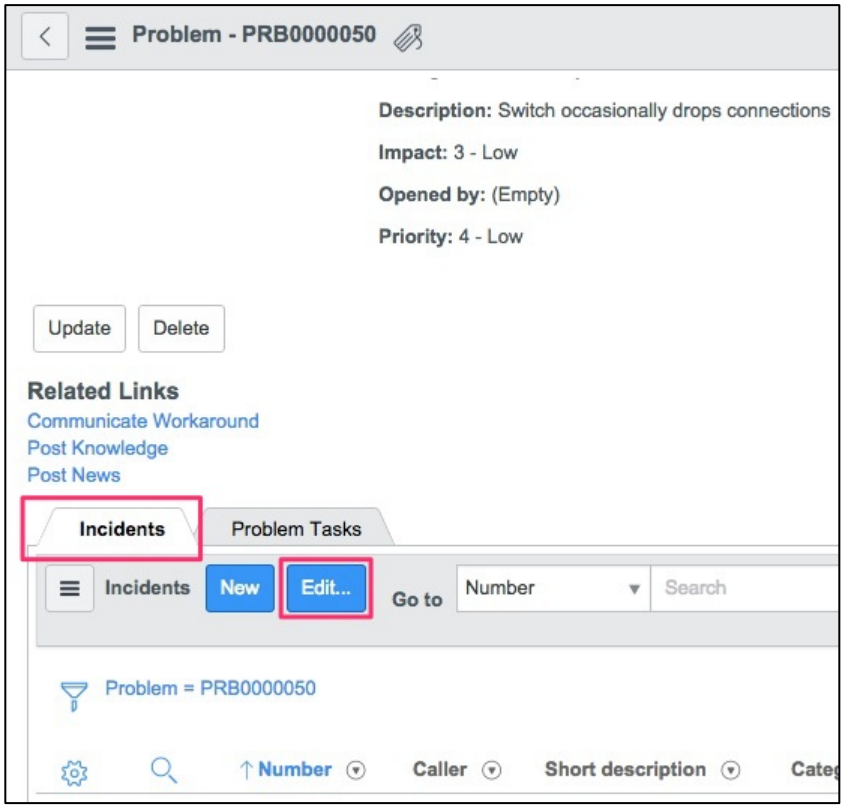


This Business Rule displays an information message showing the number of related/child Change Requests whenever the Incident is updated.

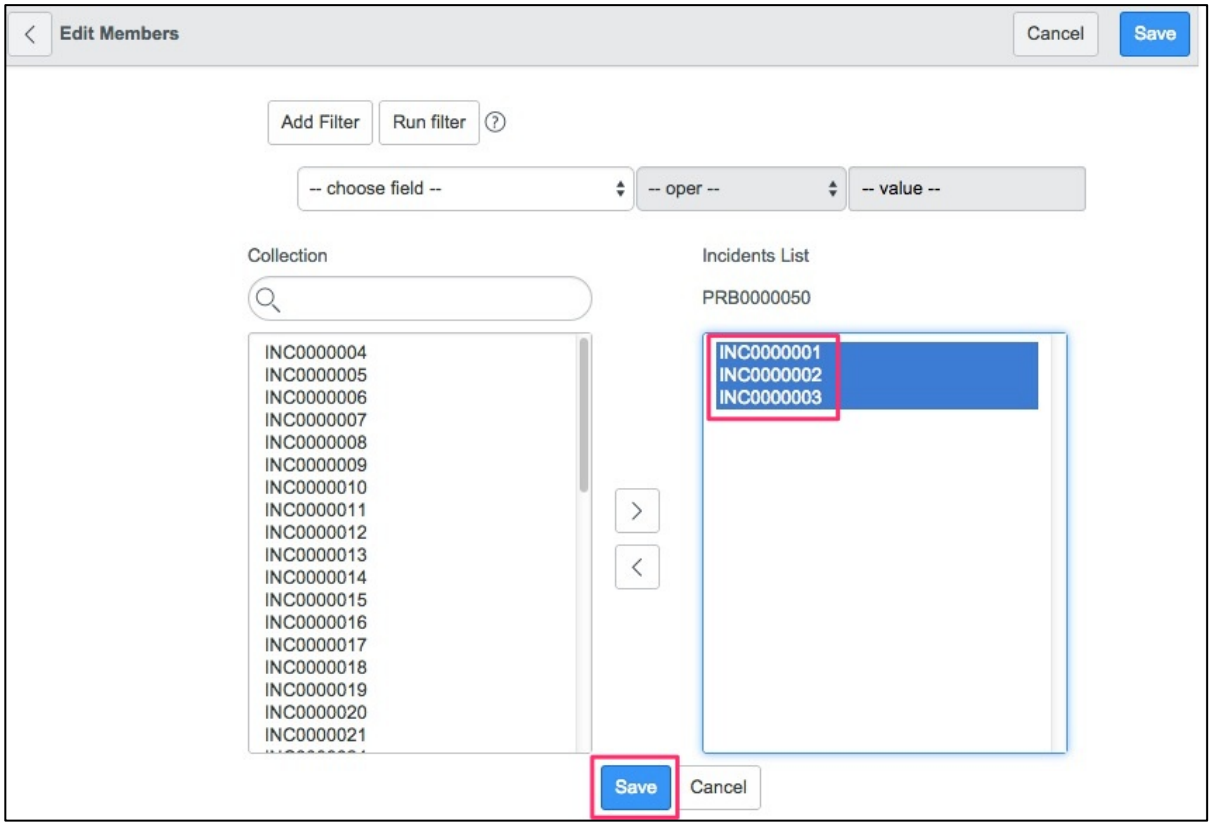
9. Test the functionality and see the Business Rules working by opening **PRB0000050**.



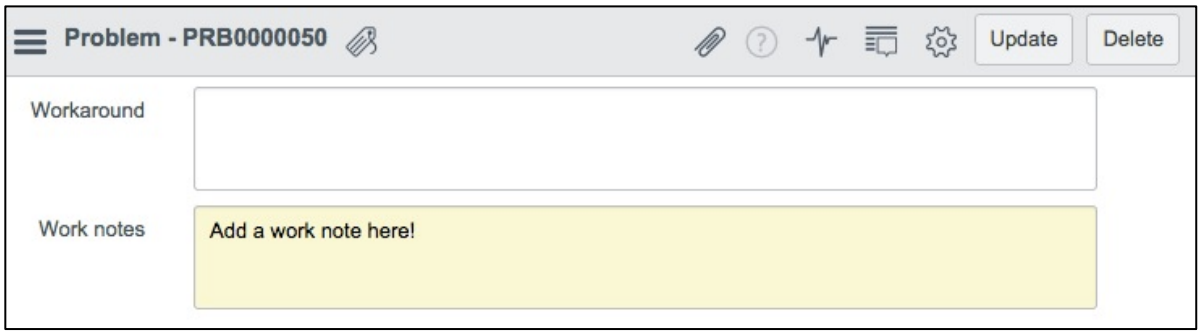
10. Click the **Incident Tab** located below the form. Edit the related incidents for **PRB0000050** by clicking the **Edit** button.



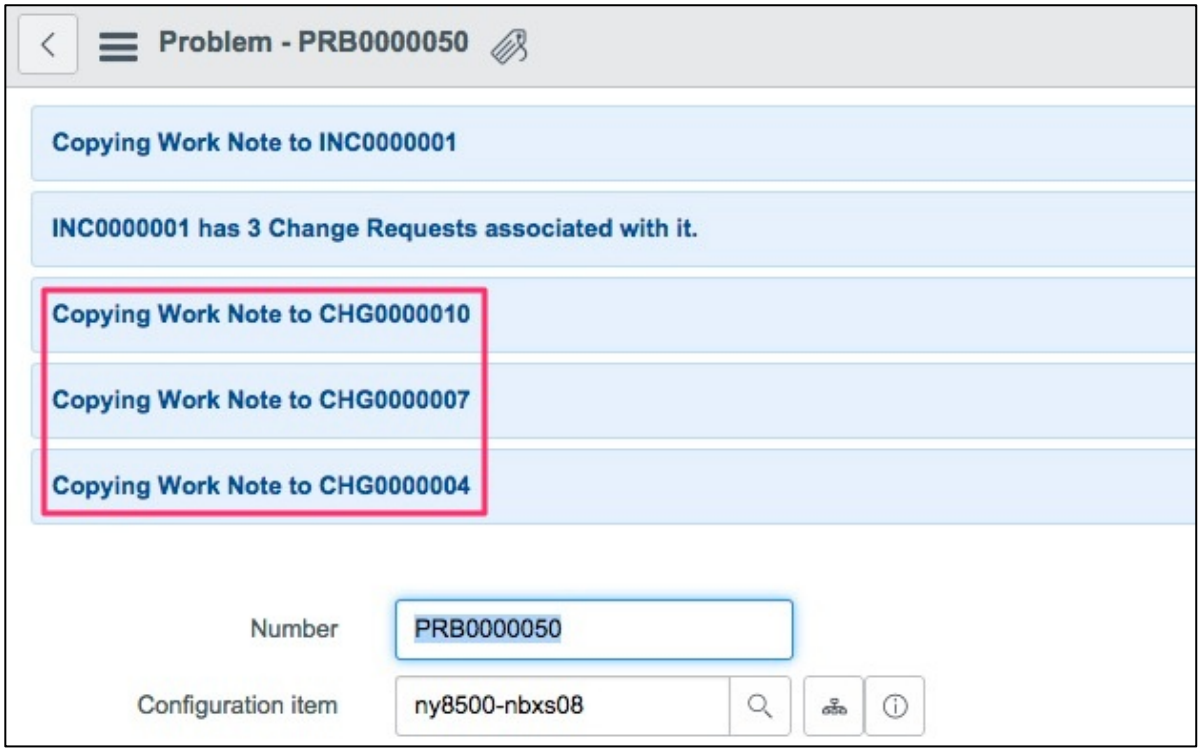
11. Add **INC0000001, INC0000002,** and **INC0000003. Save** the changes.



12. Add a **Work note** to **PRB0000050** and Save.



13. Notice that the Work note is copied to **INC0000001**. A count of the number of change requests displays, as expected. But then the work note is copied to **CHG0000010**, **CHG0000007**, and **CHG0000004**, which is not expected.



14. View each of the records affected to verify what has happened.

15. To fix the Business Rule so it works as expected return to the **Copy Work Note To Related Incidents** Business Rule. Wrap the script in an immediately invoked function**.**

```
(function() {
    var gr = new GlideRecord("incident");
    gr.addQuery("problem_id", current.sys_id);
    gr.query();

    while(gr.next()) {
        gs.addInfoMessage("Copying Work Note to " + gr.getDisplayValue());
        gr.comments = current.work_notes;
        gr.update();
    }
})();
```

16. Save the **Business Rule**.

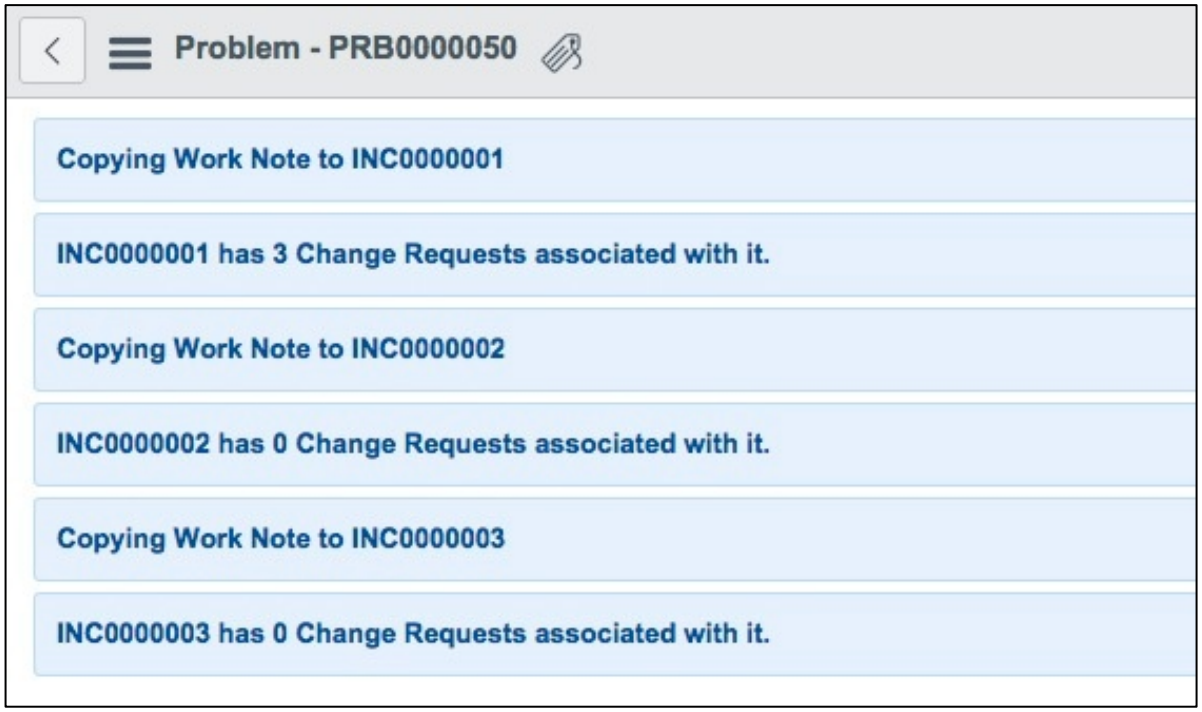
17. Return to the **Active Change Requests** Business Rule. Wrap the script in an immediately invoked function**.**

```
(function() {
    var gr = new GlideRecord("change_request");
    gr.addQuery("parent", current.sys_id);
    gr.query();

    gs.addInfoMessage(current.number + " has " + gr.getRowCount() + " Change Requests associated with it.");
})();
```

18. Save the **Business Rule**.

19. Add another **Work Note** to the **PRB0000050**.

20. Notice the result this time is as expected.



21. View each of the records affected to **verify** what has happened.

Lab Challenge

1. Create a new Advanced Before Business Rule on the Problem table, and add any script you like. What do you notice about the script field on the form? Try to remove the pre-populated function and save the Business Rule. What happens?

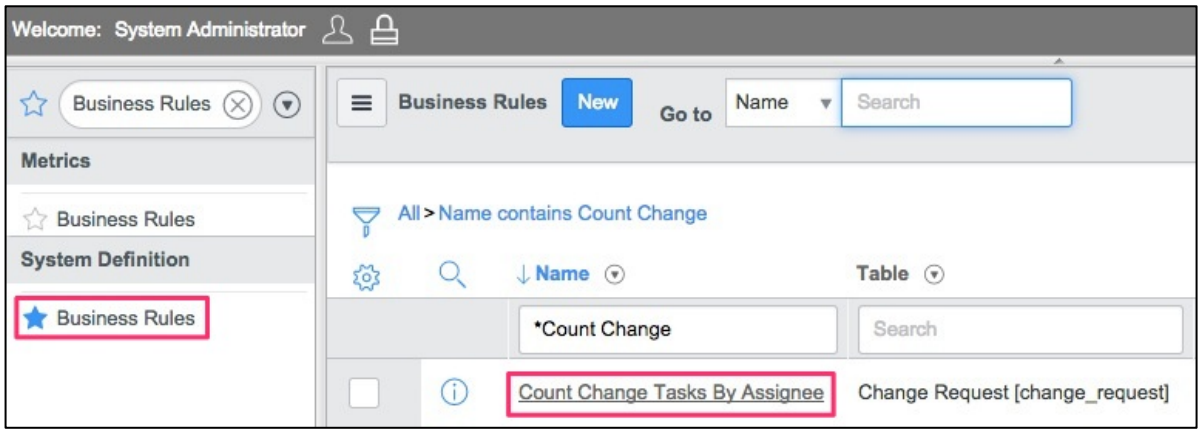
Lab Goal

This lab explains how to leverage the Server-Side JavaScript Debugger to effectively debug a faulty business rule.

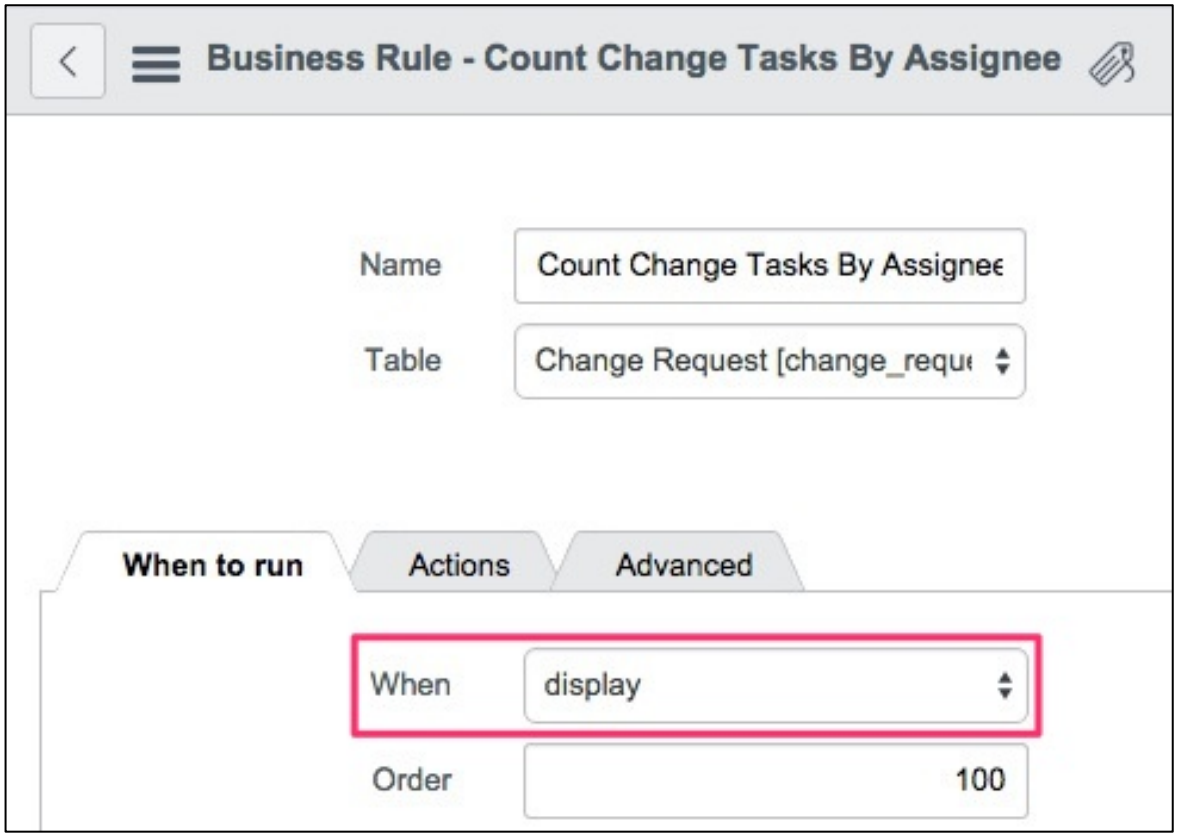
Lab 2
Server-Side JavaScript Debugger

Server-Side JavaScript Debugger

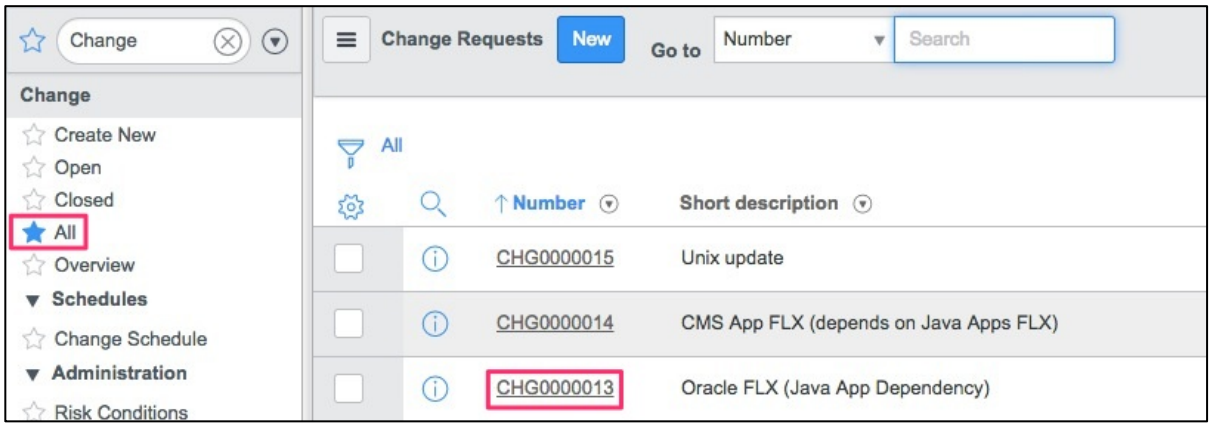
1. Navigate to **System Definition > Business Rules** and open the business rule **Count Change Tasks By Assignee**.



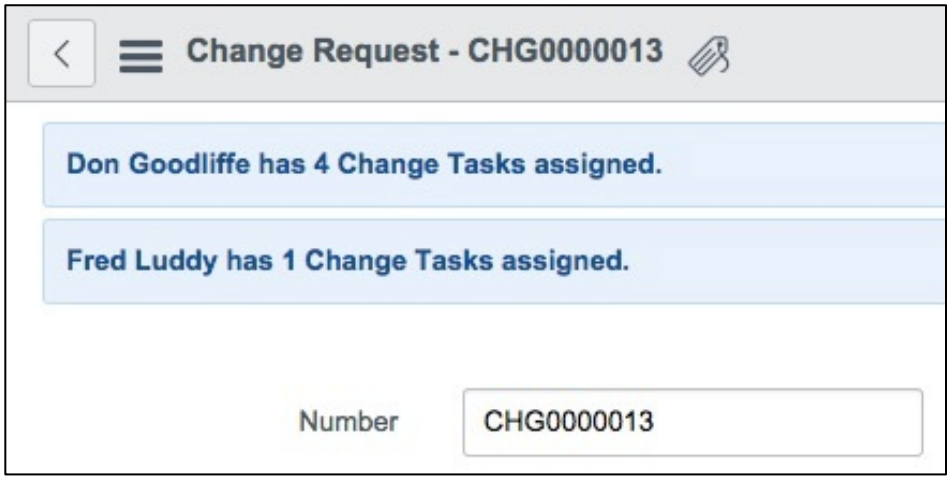
2. **Change Tasks By Assignee** is a **Display** business rule that prints the Change Task workload for engineers working on the current Change Request. It looks at the assignee for the current Change as well as the assignees for any associated Change Tasks.

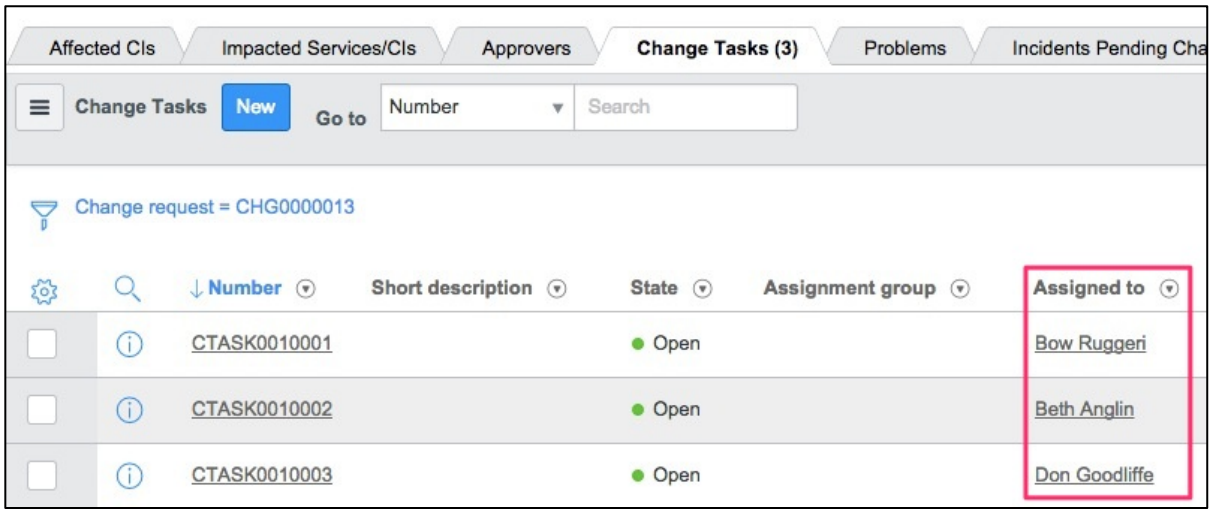


3. Navigate to **Change Requests** and open **CHG0000013**.

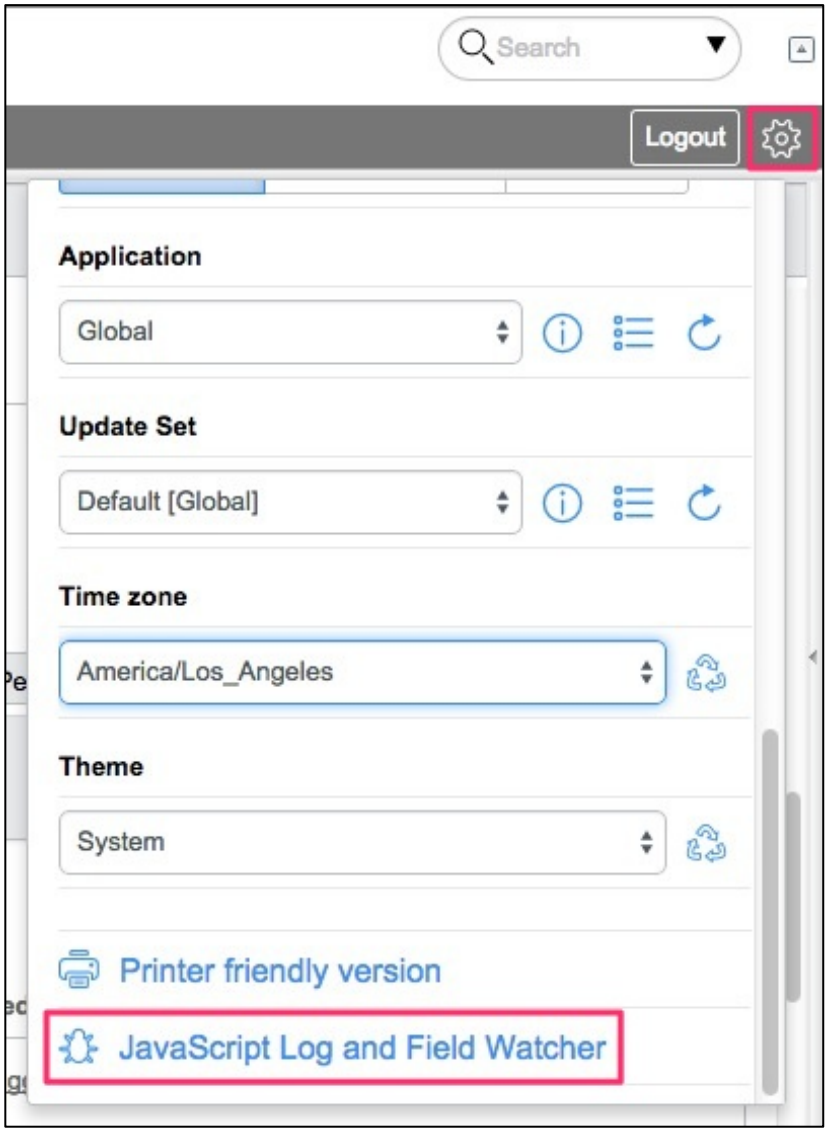


4. Change task counts *should* be displayed for the change request assignee (Fred Luddy) and each user assigned to a change task (Bow Ruggeri, Don Goodliffe, and Beth Anglin). However, only counts for Fred and Beth are reported.

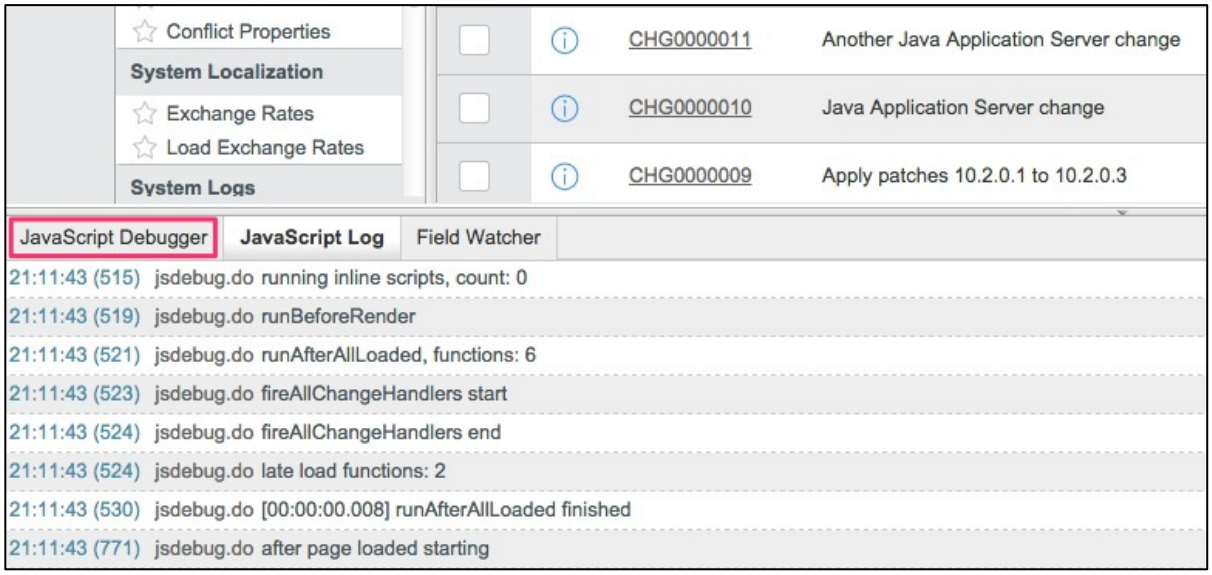




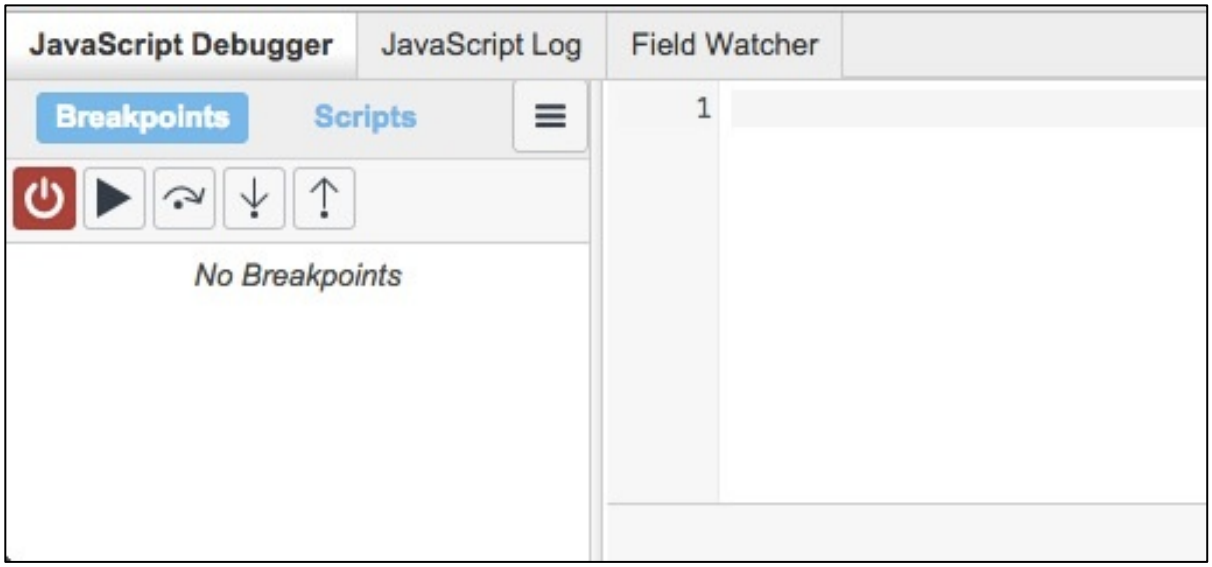
5. Click the **cog** in the upper-right corner of the UI and select **JavaScript Log and Field Watcher**.



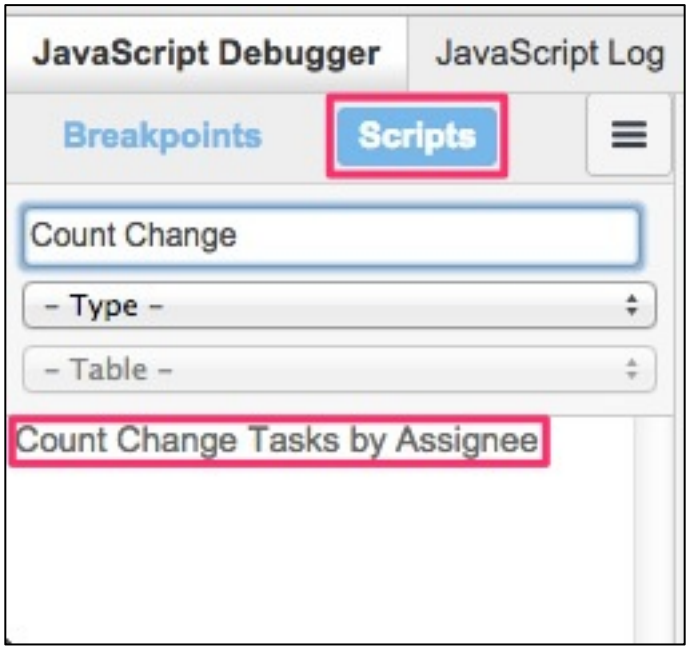
6. Click the **JavaScript Debugger** tab at the bottom-left of the screen.



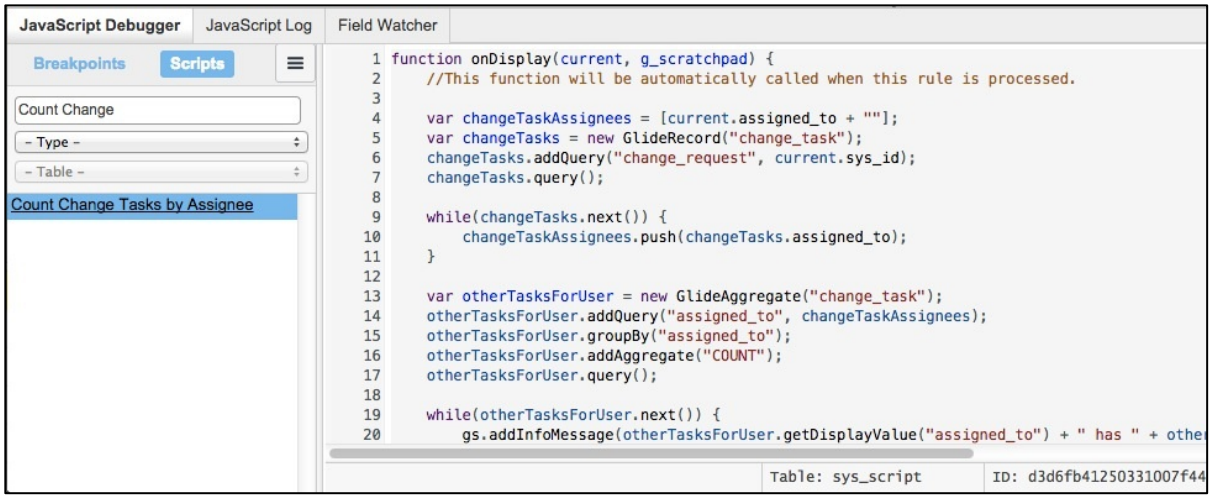
7. The **JavaScript Debugger** is now on and ready for use.



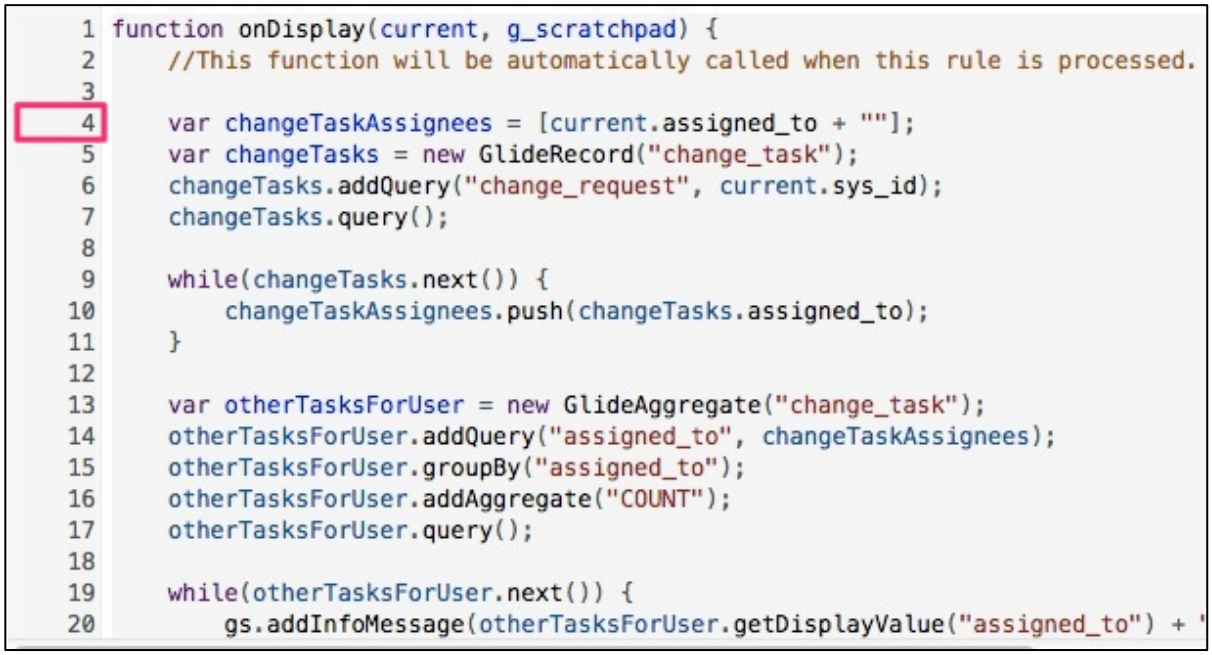
8. Click on the **Scripts** button and search for the script titled **Count Change Tasks By Assignee**.



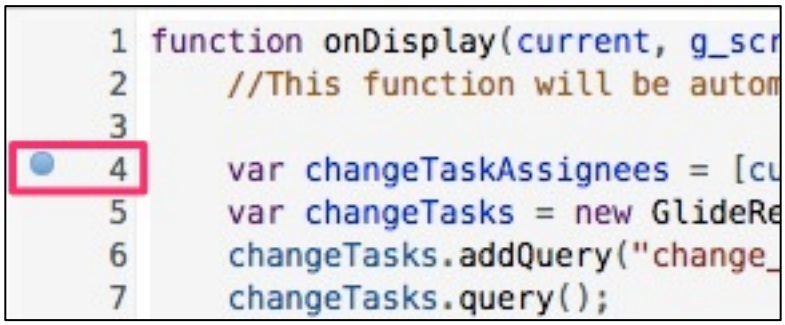
9. Clicking on the script opens it in the debugger script view.



10. Click in the gutter next to **line 4** to set a breakpoint.

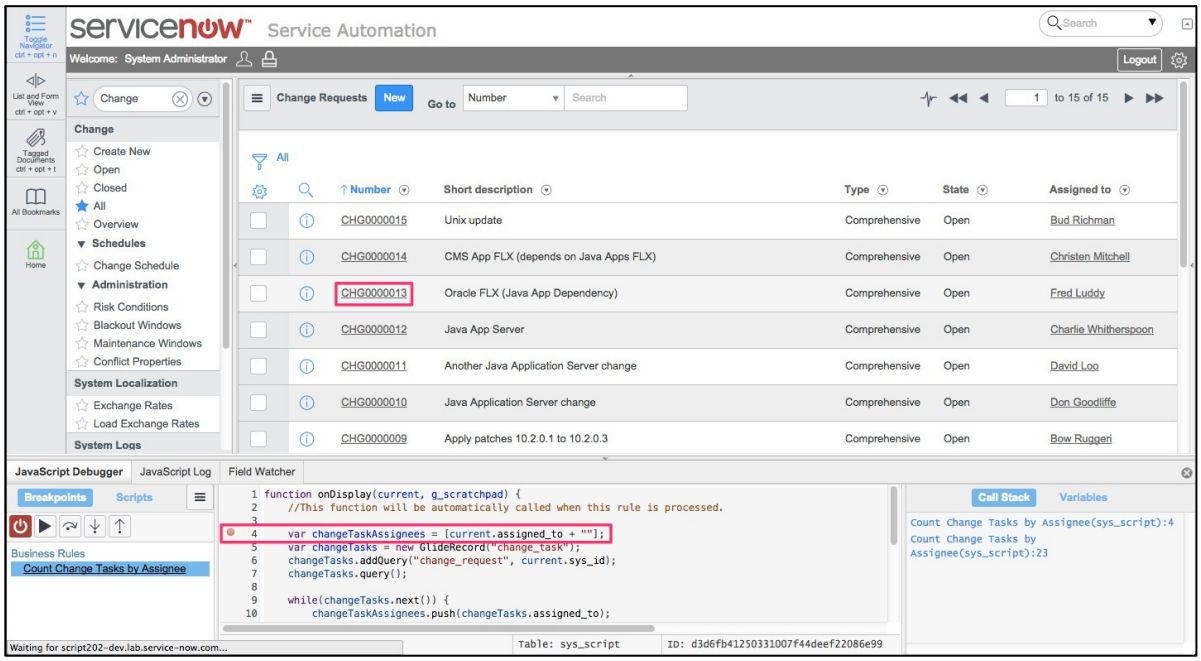


```
1  function onDisplay(current, g_scratchpad) {
2      //This function will be automatically called when this rule is processed.
3
4      var changeTaskAssignees = [current.assigned_to + ""];
5      var changeTasks = new GlideRecord("change_task");
6      changeTasks.addQuery("change_request", current.sys_id);
7      changeTasks.query();
8
9      while(changeTasks.next()) {
10         changeTaskAssignees.push(changeTasks.assigned_to);
11     }
12
13     var otherTasksForUser = new GlideAggregate("change_task");
14     otherTasksForUser.addQuery("assigned_to", changeTaskAssignees);
15     otherTasksForUser.groupBy("assigned_to");
16     otherTasksForUser.addAggregate("COUNT");
17     otherTasksForUser.query();
18
19     while(otherTasksForUser.next()) {
20         gs.addInfoMessage(otherTasksForUser.getDisplayValue("assigned_to") + '
```

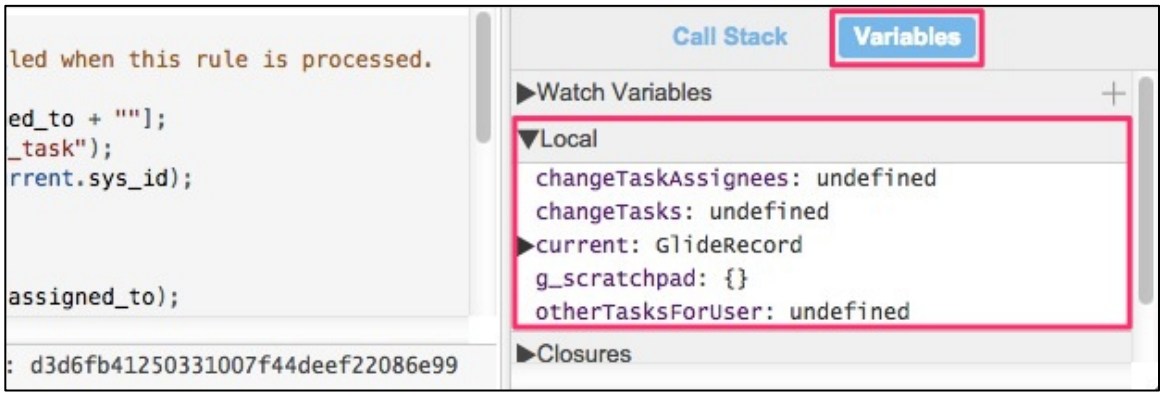


```
1  function onDisplay(current, g_scr
2      //This function will be autom
3
4      var changeTaskAssignees = [cu
5      var changeTasks = new GlideRe
6      changeTasks.addQuery("change_
7      changeTasks.query();
```

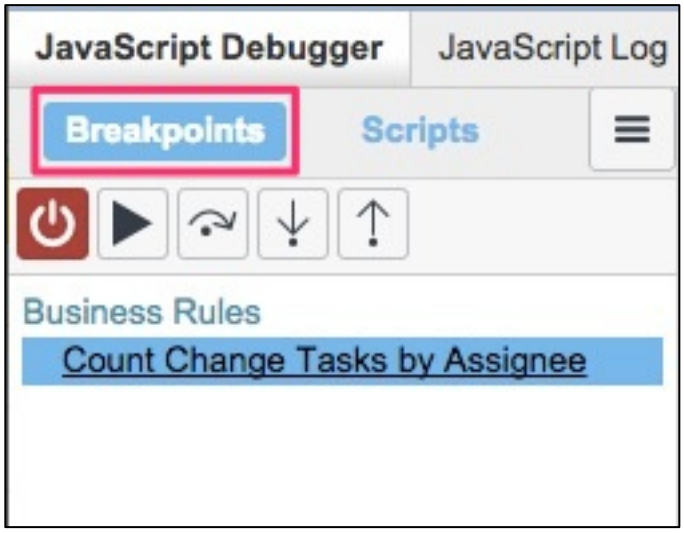
11. View **CHG0000013** and notice that execution has paused on the second line. A **red breakpoint** indicates that execution has paused on the breakpoint.



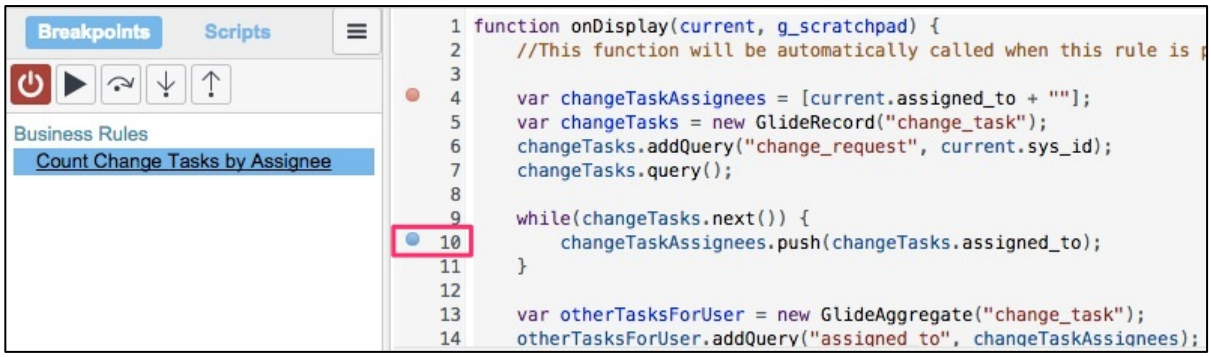
12. Click the **Variables** button on the right side of the debugger and open the **Local** section to see variables local to the current scope.



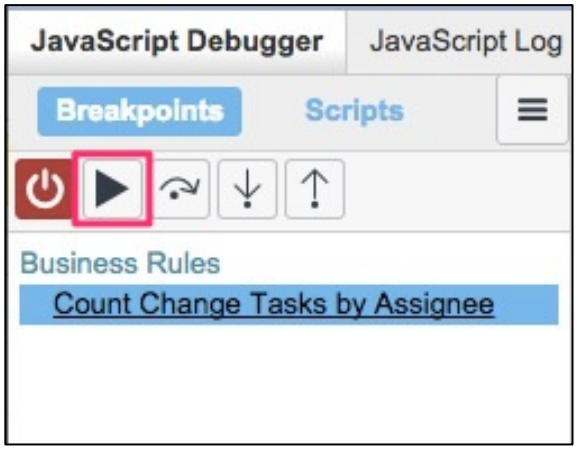
13. Click the **Breakpoints** button.



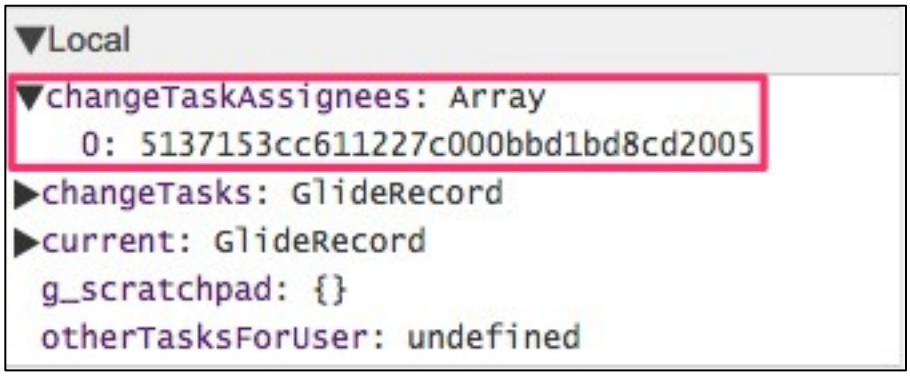
14. Add a breakpoint on **line 10.**



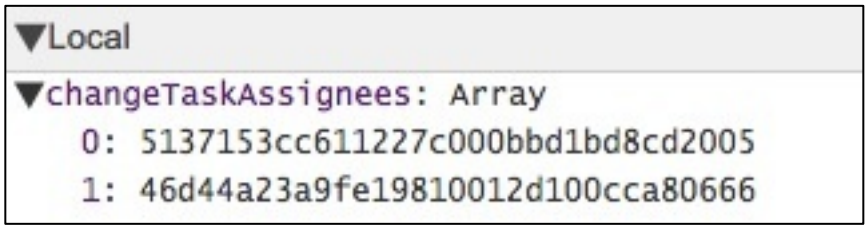
15. Press the **Play** button to advance to the next breakpoint.



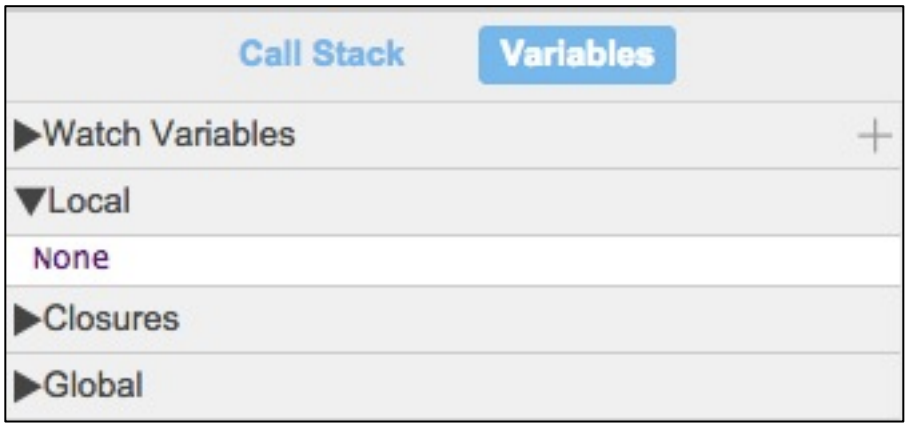
16. Expand the **changeTaskAssignees** variable to see what values it contains.



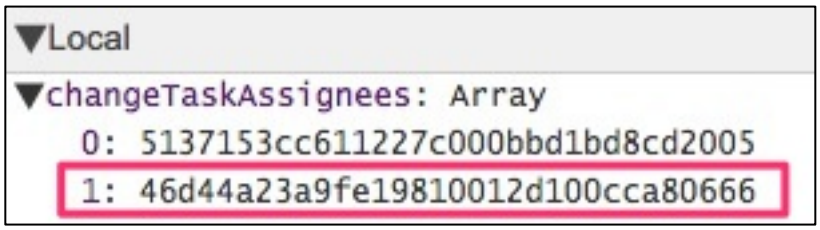
17. Click the **Play** button to continue execution. The debugger pauses on **line 10** again because it is iterating over a GlideRecord collection with multiple entries. It pauses on **line 10** once per change task. Notice that values in the **changeTaskAssignees** are updated as the script iterates over **changeTasks.**

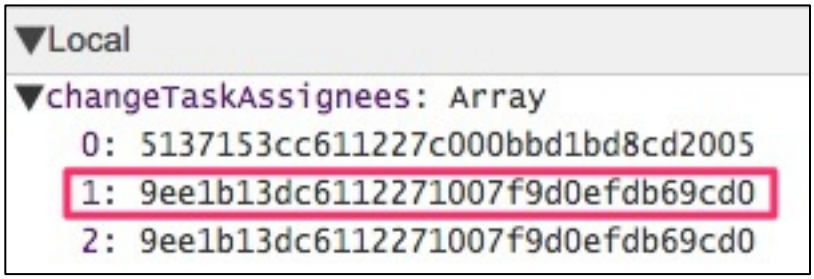


18. Continue pressing **Play** until execution completes. Local variables disappear when execution completes.



19. Open the same change request and step through execution again. Pay close attention to the values in **changeTaskAssignees.** Notice that each value (except the first) updates **on every iteration**. This behavior is wrong. Existing values in the array should not change.



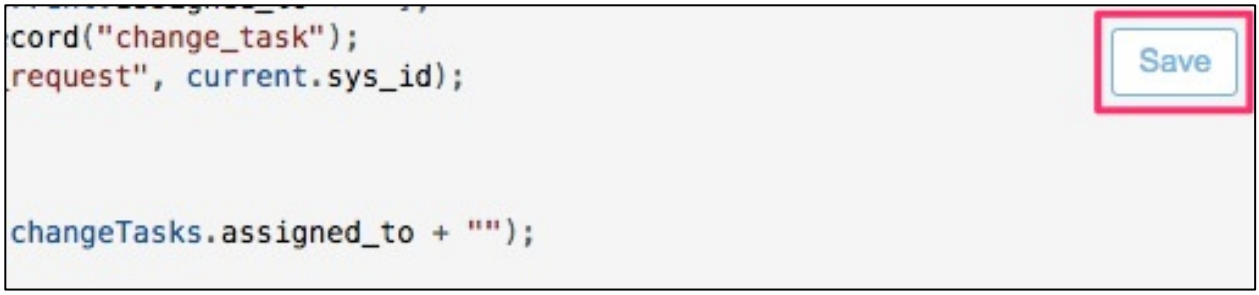


20. Change **line 10** to de-reference the **assigned_to** value before adding it to **changeTaskAssignees**. This can be done within the debugger interface.

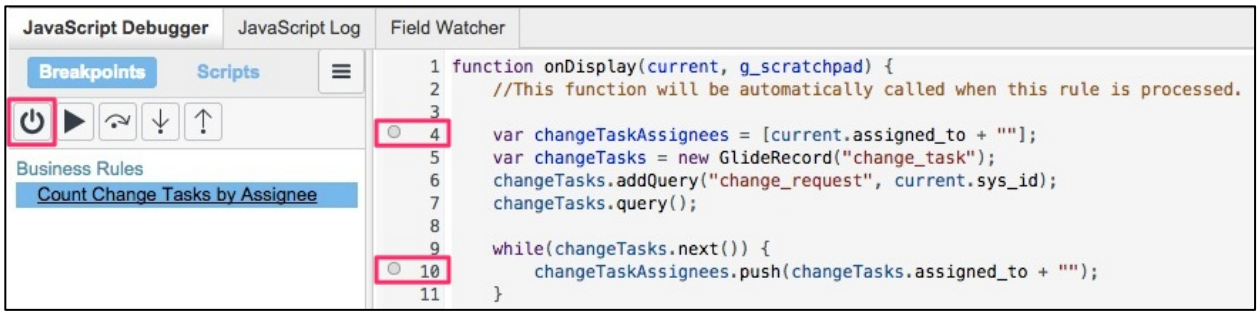
```
 9    while(changeTasks.next()) {
10        changeTaskAssignees.push(changeTasks.assigned_to);
11    }
```

```
 9    while(changeTasks.next()) {
10        changeTaskAssignees.push(changeTasks.assigned_to + "");
11    }
```

21. Click the **Save** button in the debugger's script view to save **Count Change Tasks By Assignee**.



22. Turn the debugger off by clicking on the **ServiceNow symbol.** It changes from red to gray to indicate that the debugger is off. Notice too that the breakpoints on **line 4** and **line 10** are grayed out.



23. Navigate to **CHG0000013.** Notice that counts now display for all assignees.

Lab Success Verification

1. Navigate to **CHG0000013**. Notice that it displays Change Task counts for all users assigned to the **CHG0000013** change tasks.

Note: There is a bug in the script that means you may see two lots of messages.

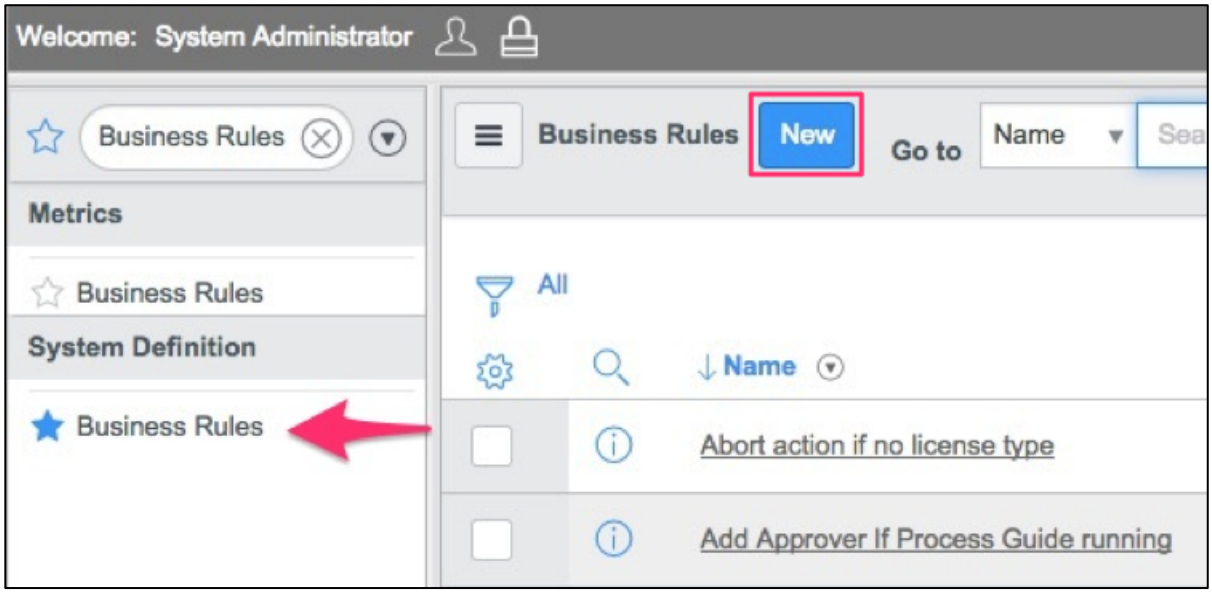
Lab Goal

This lab explains how to use Script Includes to eliminate duplicate code and centralize business rule conditions.

Lab 3.1
Do Not
Repeat
Yourself

Consolidating Common Code Using Script Includes

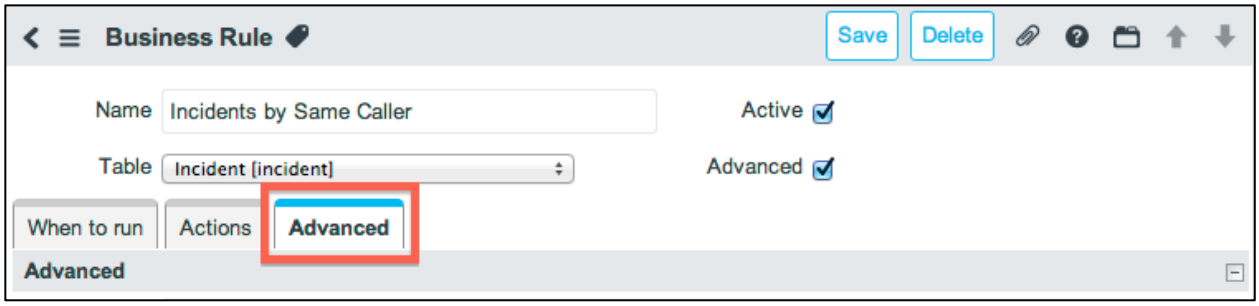
1. Navigate to **System Definition > Business Rules** and click **New**.



2. Create a new **Advanced Rule** called **Incidents by Same Caller** and configure it to run against **Incident.**

3. Configure Incidents by **Same Caller** to run on Display.

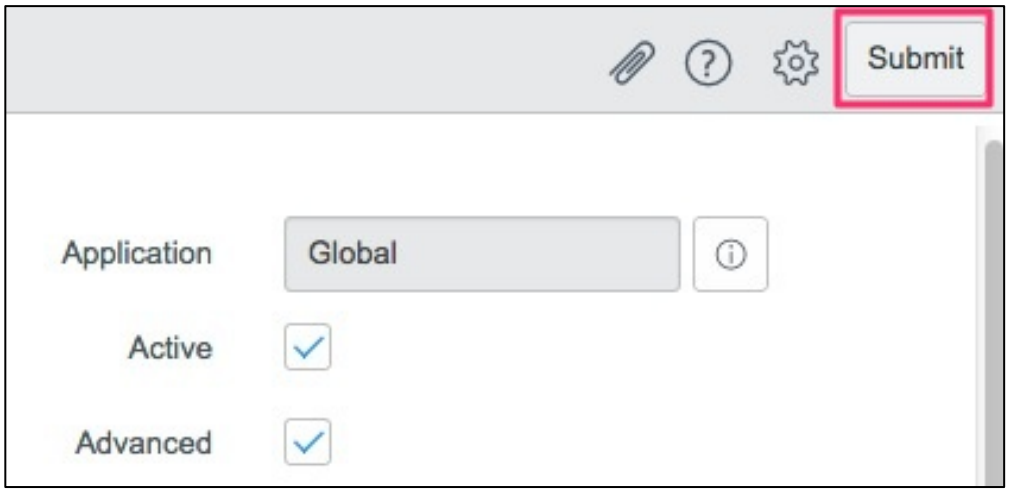
4. Navigate to the **Advanced** section of the business rule.



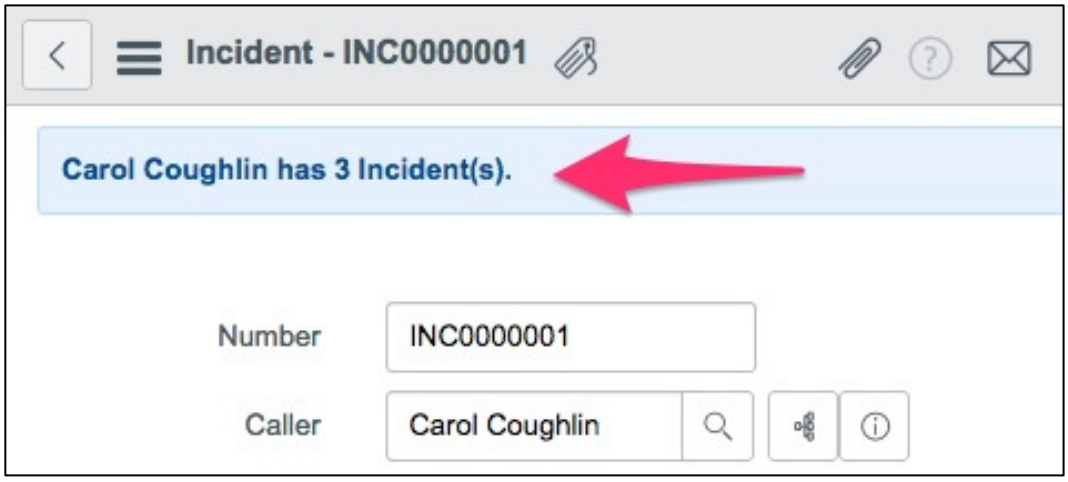
5. Write a script for **Incidents by Same Caller** that queries Incident for records where **caller_id** equals **current.caller_id**. Use **gs.addInfoMessage** to display the result on screen.

```
function onDisplay(current, g_scratchpad) {
  var incidentsBySameCaller = new GlideRecord("incident");
  incidentsBySameCaller.addQuery("caller_id", current.caller_id);
  incidentsBySameCaller.query();

  gs.addInfoMessage(current.caller_id.getDisplayValue() + " has " +
incidentsBySameCaller.getRowCount() +  " Incident(s).");
}
```

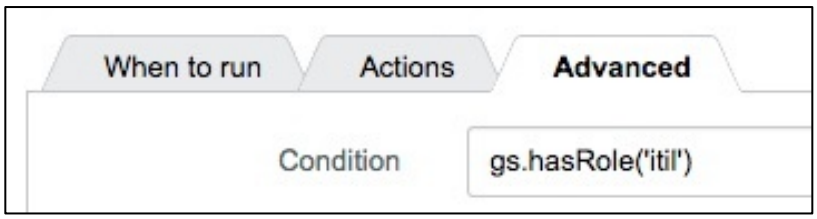
6. Save Incidents by **Same Caller** by clicking **Submit**.



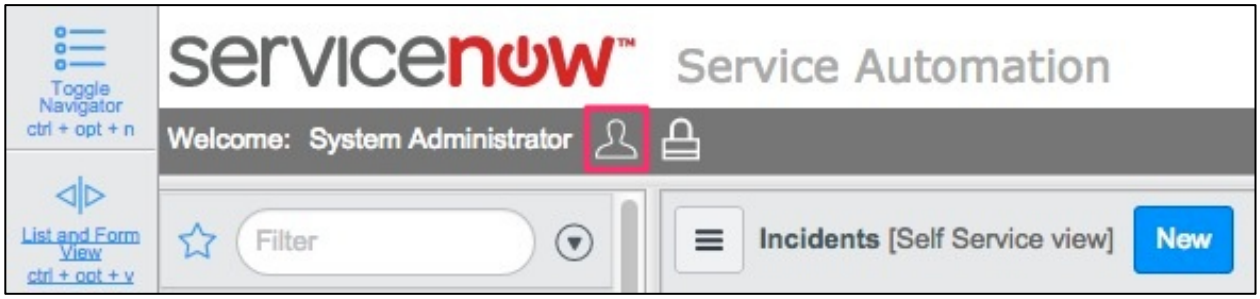
7. Open **INC0000001** and confirm that it shows three incidents by the same user.

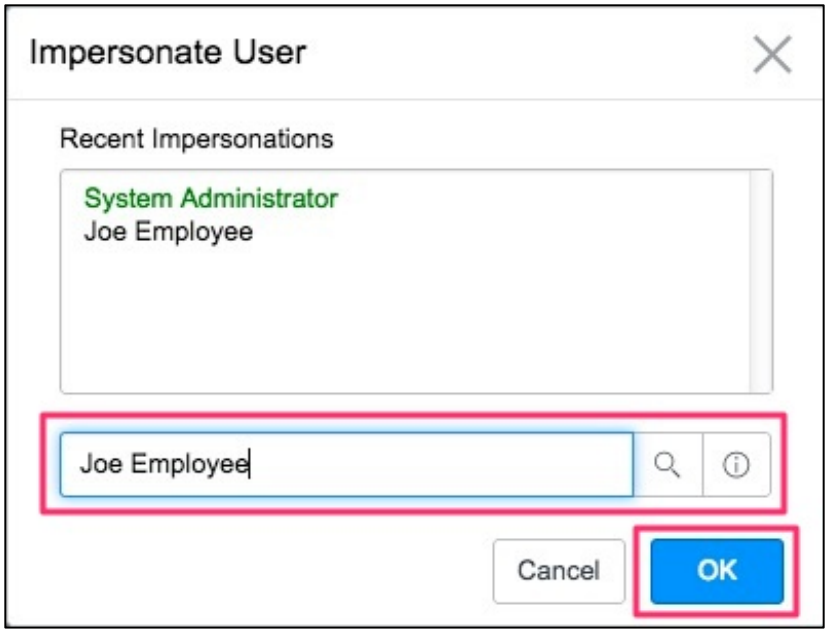


8. You want this rule to only run for users with the **ITIL** role, not all users. Add a condition to **Incidents by Same Caller** so that it runs a role check. Use **`gs.hasRole('itil')`**

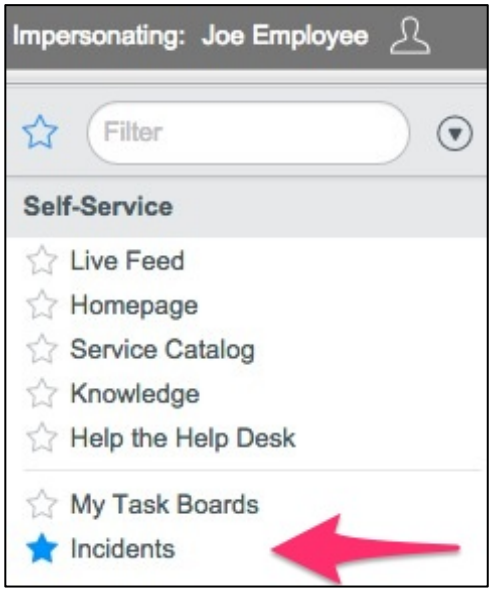


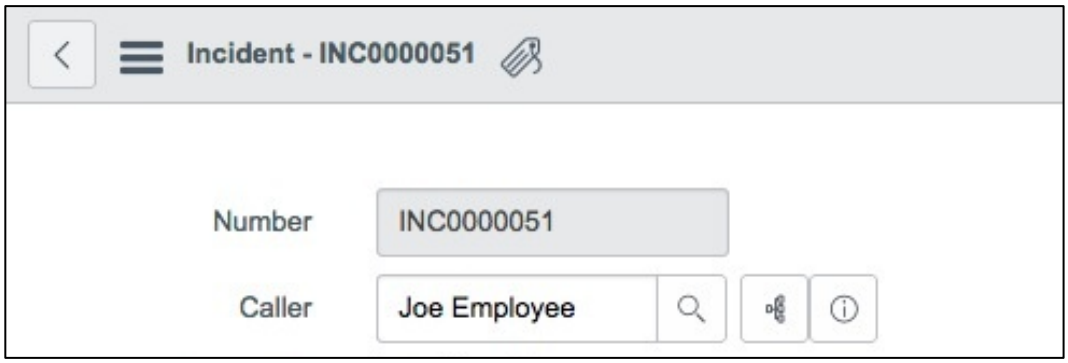
9. Impersonate **Joe Employee** by clicking on the user icon in the upper-left screen.





10. Open one of Joe Employees incidents and confirm that no incident counts are displayed.





11. Use the impersonation utility again to resume running as **System Administrator**.

12. You now want to be able to perform similar analysis on Problem. Navigate back to **Incidents by Same Caller**. Right-click the header for **Incidents by Same Caller** and select **Insert and Stay** from the context menu that appears. This duplicates the rule.

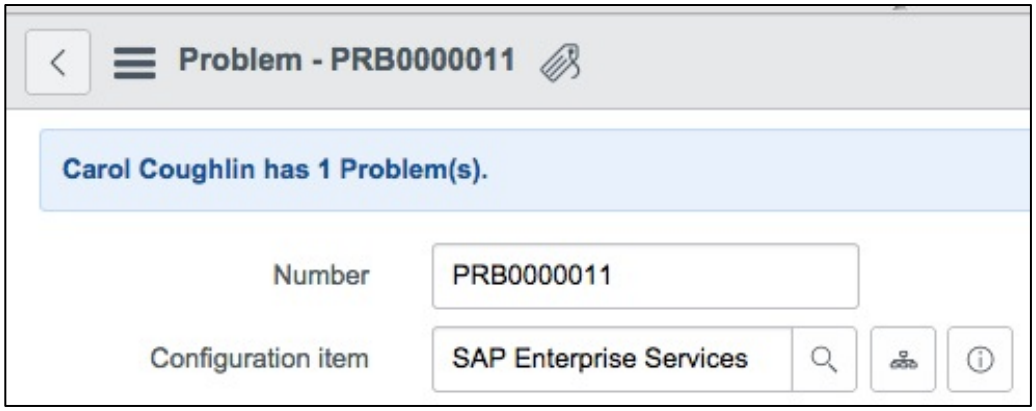
13. Change the title to **Problems Opened by Same User** and change the table to **Problem.**

14. Change the script to query **Problem**. The field being filtered is **opened_by**.

```
function onDisplay(current, g_scratchpad) {
  var problemsBySameUser = new GlideRecord("problem");
  problemsBySameUser.addQuery("opened_by", current.opened_by);
  problemsBySameUser.query();

  gs.addInfoMessage(current.opened_by.getDisplayValue() + " has " +
problemsBySameUser.getRowCount() +  " Problem(s).");
}
```

15. Save **Problems Opened by Same User**.

16. Open **PRB0000011** and you should see the following:

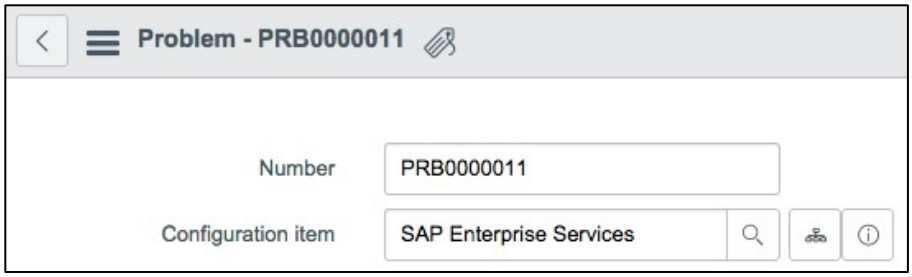


17. Add an additional condition to **Problems Opened by Same User** and **Incidents by Same Caller** that checks if the target user's company is a critical account. The **core_company** table has a field called **u_critical_account.**

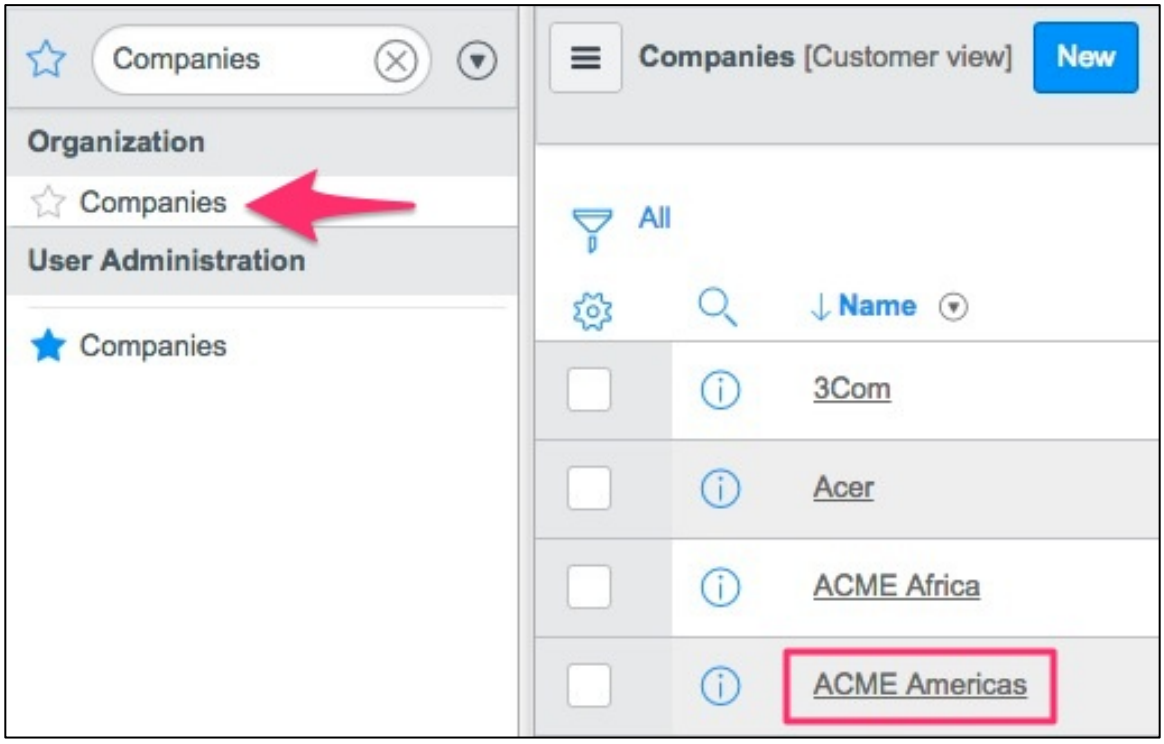
Incident:

Problem:

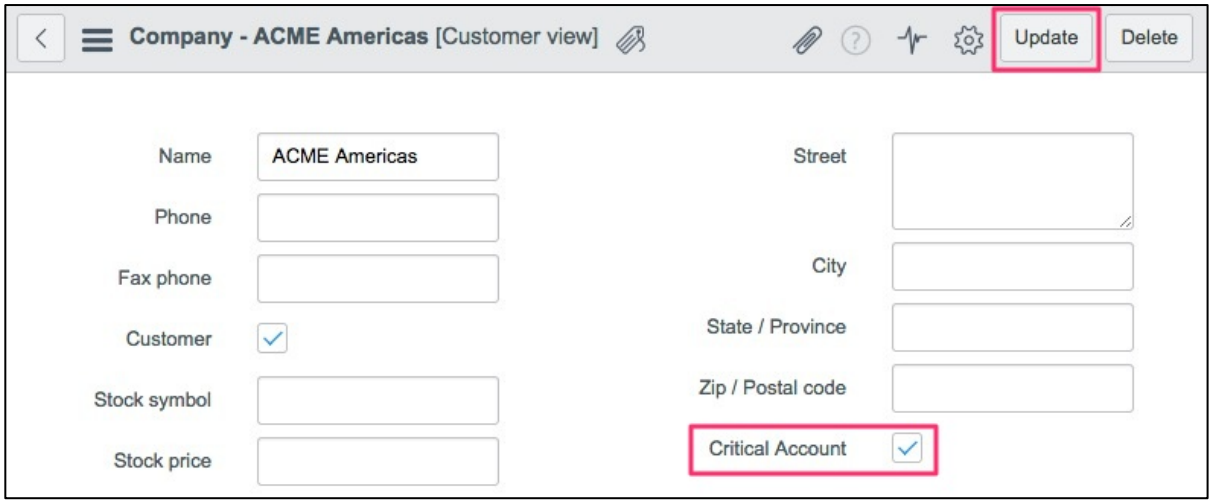
18. Open **PRB0000011** and confirm that PRB counts for **Carol Coughlin** are not visible.



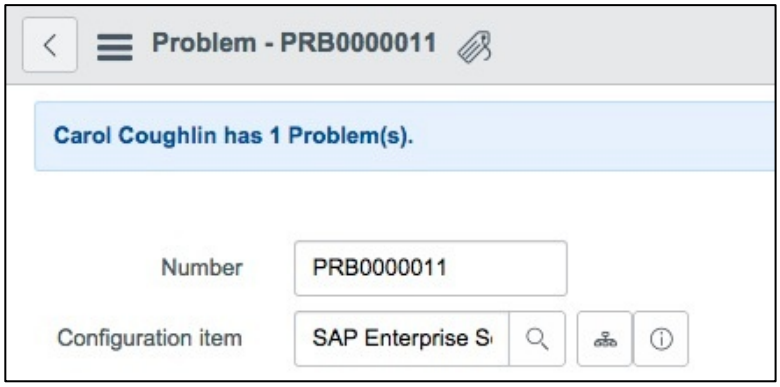
19. Open the **Company** record for ACME Americas



20. Toggle the **Critical Account** field to **true** and click **Update** to save**.**



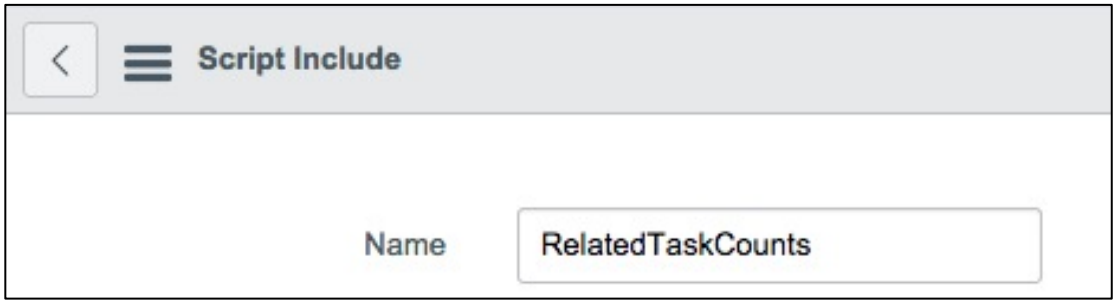
21. Open **PRB0000011** and confirm that PRB counts for **Carol Coughlin** *are* visible.



22. Update both rules to only query **active** records. For example, the rule on Problem should be as follows:

```
function onDisplay(current, g_scratchpad) {
  var problemsBySameUser = new GlideRecord("problem");
  problemsBySameUser.addQuery("opened_by", current.opened_by);
  problemsBySameUser.addActiveQuery();
  problemsBySameUser.query();

  gs.addInfoMessage(current.opened_by.getDisplayValue() + " has " +
problemsBySameUser.getRowCount() +  " Problem(s).");
}
```

23. Navigate to **System Definition > Script Includes**. Create a new Script Include called **RelatedTaskCounts**.



24. Add a method called **checkConditions** that handles condition checks for both business rules. Remember to make the method generic so that it can be applied to any child of **Task**. Save **RelatedTaskCounts**.

```
var RelatedTaskCounts = Class.create();
RelatedTaskCounts.prototype = {
    initialize: function() {
    },

  checkConditions: function(current, field) {
        return gs.hasRole('itil') &&
              current[field].company.u_critical_account;
  },

    type: 'RelatedTaskCounts'
};
```

25. Update both business rules to use the use the newly created Script Include. Remember to change the second argument (**opened_by** or **caller_id**) based on the target table (**problem** or **incident**).

Problem:

Condition	new RelatedTaskCounts().checkConditions(current, "opened_by")

Incident:

Condition	new RelatedTaskCounts().checkConditions(current, "caller_id")

26. Add a method called **getRecordCount**. This method performs the same function as either business rule. It takes a record and a field name as arguments. Save **RelatedTaskCounts.**

```
var RelatedTaskCounts = Class.create();
RelatedTaskCounts.prototype = {
    initialize: function() {
    },

  checkConditions: function(current, field) {
        return gs.hasRole('itil') &&
              current[field].company.u_critical_account;
  },

  getRecordCount: function(current, field) {
        var recordsBySameUser = new GlideRecord(current.getTableName());
        recordsBySameUser.addQuery(field, current[field]);
        recordsBySameUser.addActiveQuery();
        recordsBySameUser.query();

        gs.addInfoMessage(current[field].getDisplayValue() + " has " +
recordsBySameUser.getRowCount() +  " active " +
current.getClassDisplayValue() + "(s) .");
    },

    type: 'RelatedTaskCounts'
};
```

27. Update both business rules to leverage **RelatedTaskCounts**.

Problems Opened by Same User:

```
function onDisplay(current, g_scratchpad) {
  new RelatedTaskCounts().getRecordCount(current, "opened_by");
}
```

Incidents by Same Caller:

```
function onDisplay(current, g_scratchpad) {
  new RelatedTaskCounts().getRecordCount(current, "caller_id");
}
```

28. Update **RelatedTaskCounts** to apply only if the user is a VIP: change the **checkConditions** method. Notice that you now only need to make this change in one location.

```
  checkConditions: function(current, field) {
        return gs.hasRole('itil') &&
              current[field].company.u_critical_account &&
              current[field].vip;
  },
```

Lab Success Verification

1. Open incident counts should display for all VIP users from critical accounts. You should be able to update these criteria in a single place and see the criteria applied to Incident and Problem records.

   Open **INC0000050**. You should see the following:

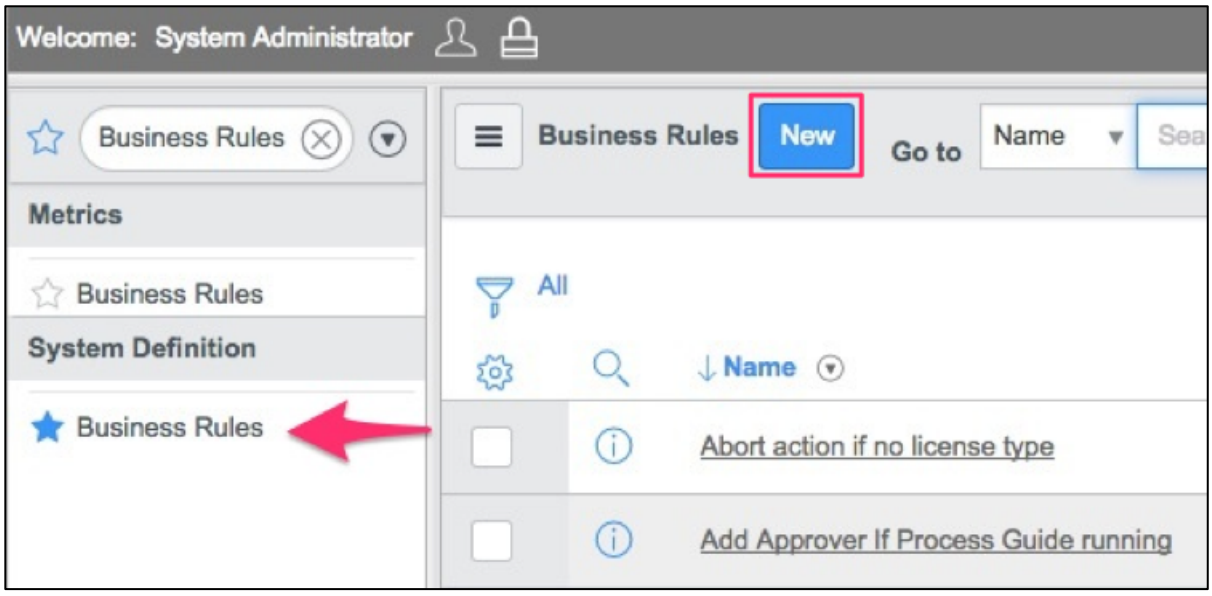
Lab Goal

This lab explores decomposing complex code to extract common routines. You learn to move record counting code into its own script include so that it can be re-used for incident reassignment.

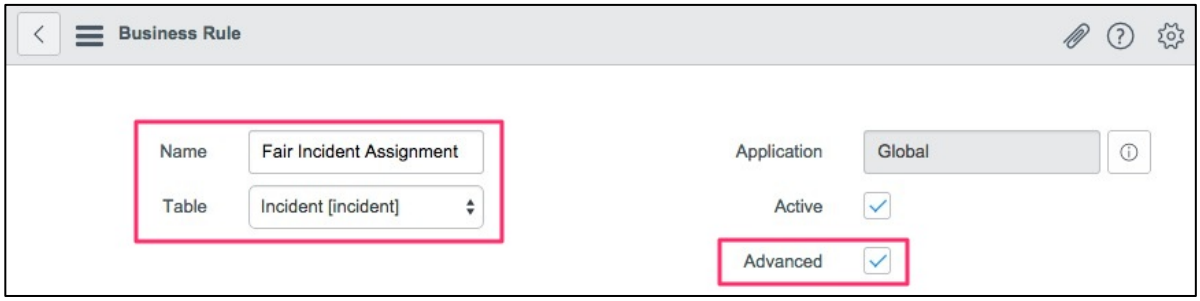
Lab 3.2
Code Reusability

Consolidate Common Code Using Script Includes

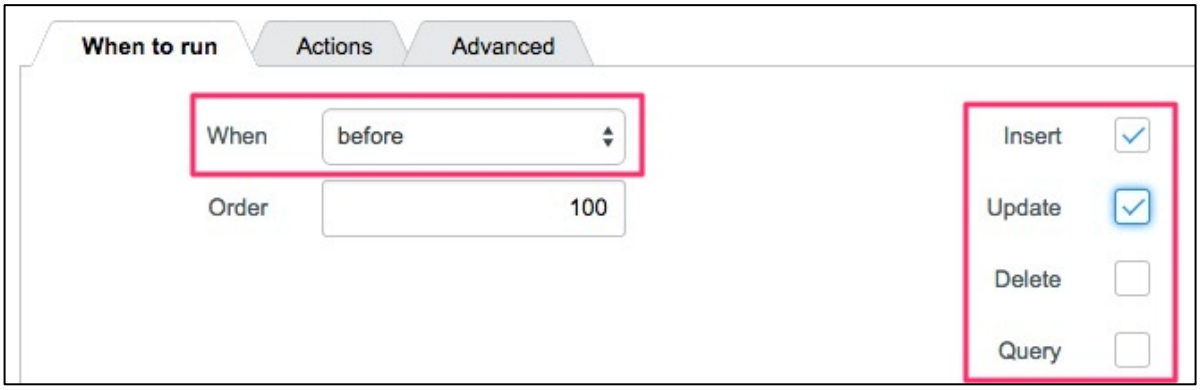
1. Navigate to **System Definition > Business Rules** and click **New**.



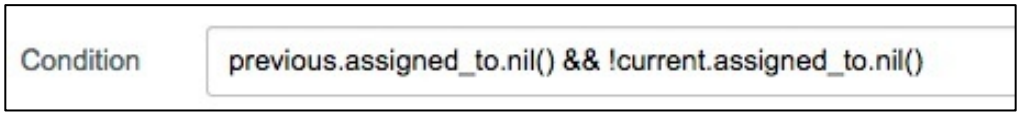
2. Complete the form as follows:
   - Name: **Fair Incident Assignment**
   - Table: **Incident**
   - Advanced: **checked**



3. Configure the rule to run **Before Update** and **Before Insert**.



4. Set a condition so that the rule is only applied if an incident moves from **Unassigned** to **Assigned**.

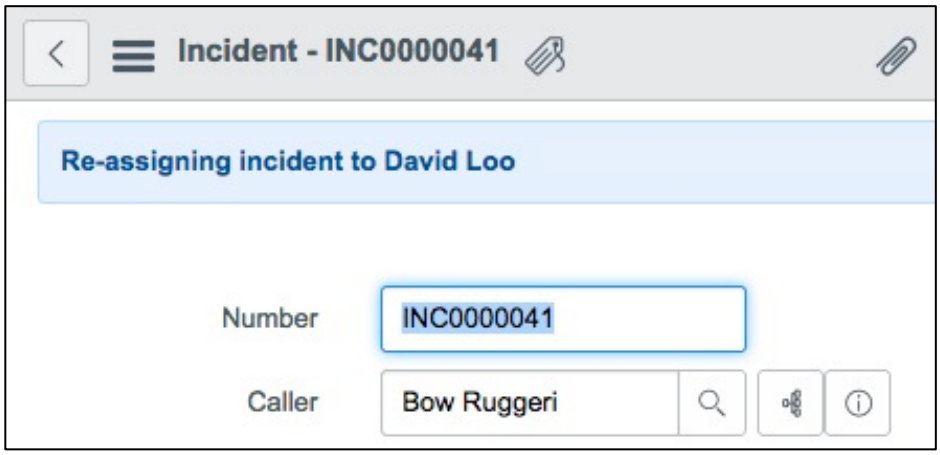


5. The **Fair Incident Assignment** Business Rule script below assigns an incident to another group member if the current assignee has more than three incidents and an engineer with fewer than three is found.

```
function onBefore(current, previous) {
  var currentUserCount = incidentCountForUser(current.assigned_to);
  if(currentUserCount > 3) {
        current.assigned_to = findUserWithLessThanThree();
  }

  function findUserWithLessThanThree() {
        var usersInThisGroup = new GlideRecord("sys_user_grmember");
        usersInThisGroup.addQuery("group", current.assignment_group);
        usersInThisGroup.query();

        while(usersInThisGroup.next()) {
              var user = usersInThisGroup.user + "";
              if(incidentCountForUser(user) < 3) {
                    gs.addInfoMessage("Re-assigning incident to " +
                    usersInThisGroup.user.getDisplayValue());
                    return user;
              }
        }
        return current.assigned_to + "";
  }

  function incidentCountForUser(user) {
        var incidents = new GlideRecord("incident");
        incidents.addQuery("assigned_to", user);
        incidents.addActiveQuery();
        incidents.query();
        return incidents.getRowCount();
  }
}
```

6. Save **Fair Incident Assignment**.

7. Open **INC0000041** and attempt to assign it to **Beth Anglin**. You should see **INC0000041 is** assigned to **someone else** instead.



8. The routine for checking incident counts is common to both **Fair Incident Assignment** and **RelatedTaskCounts**. Refactor it into a **Script Include** that can be called from both places.

9. Create a new Script Include called **RecordCounter** that contains common record counting functionality.

```
var RecordCounter  = Class.create();
RecordCounter .prototype = {
    initialize: function() {
    },

  activeRecordsByField: function(table, field, value) {
        var recordsByField = new GlideRecord(table);
        recordsByField.addQuery(field, value);
        recordsByField.addActiveQuery();
        recordsByField.query();

        return recordsByField.getRowCount();
    },

    type: 'RecordCounter '
};
```

10. Save **RecordCounter**.

11. Update **Fair Incident Assignment** to leverage **RecordCounter**.

```
function onBefore(current, previous) {
   var currentUserCount = new RecordCounter().activeRecordsByField("incident",
"assigned_to", current.assigned_to+"");
   if(currentUserCount > 3) {
          current.assigned_to = findUserWithLessThanThree();
   }

   function findUserWithLessThanThree() {
          var usersInThisGroup = new GlideRecord("sys_user_grmember");
          usersInThisGroup.addQuery("group", current.assignment_group);
          usersInThisGroup.query();

          while(usersInThisGroup.next()) {
                 var user = usersInThisGroup.user + "";
                 var recordCount = new
RecordCounter().activeRecordsByField("incident", "assigned_to", user);
                 if(recordCount < 3) {
                        gs.addInfoMessage("Re-assigning incident to " +
                        usersInThisGroup.user.getDisplayValue());
                        return user;
                 }
          }
          return current.assigned_to + "";
   }
}
```

12. Update **RelatedTaskCounts** to leverage **RecordCounter**.

```
var RelatedTaskCounts = Class.create();
RelatedTaskCounts.prototype = {
    initialize: function() {
    },

  checkConditions: function(current, field) {
          return gs.hasRole('itil') &&
                 current[field].u_critical_account &&
                 current[field].vip;
  },

  getRecordCount: function(current, field) {
        var count = new RecordCounter().activeRecordsByField(current.sys_class_name,
field, current[field]);

        gs.addInfoMessage(current[field].getDisplayValue() + " has " +
recordsBySameUser.getRowCount() +  " active " + current.getClassDisplayValue() + "(s)
.");
    },

    type: 'RelatedTaskCounts'
};
```

Lab Success Verification

1. All behavior in Lab 3.1 should continue to function.

2. Any incident assigned to an overloaded engineer should be reassigned to an engineer with a smaller workload. Assigning an incident to an overloaded user should result in it being reassigned.

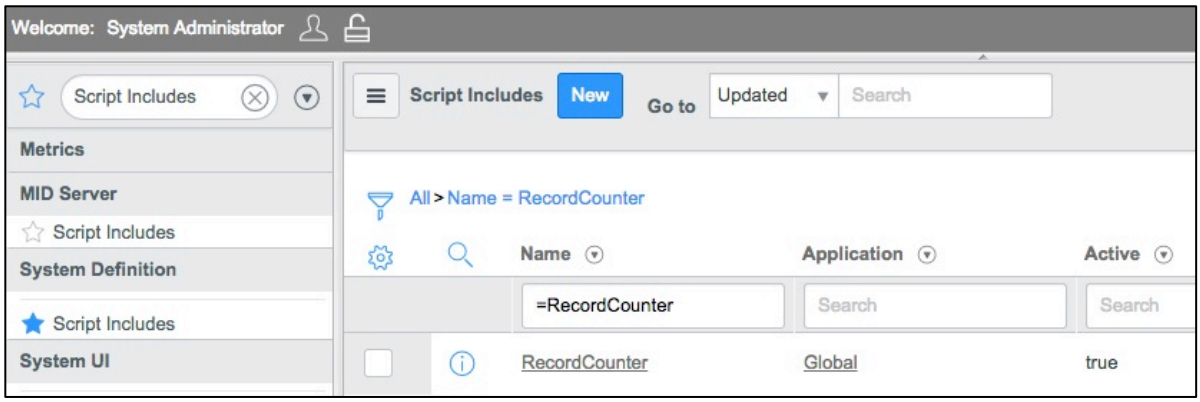
Lab Goal

This lab illlustrates the benefits of modular code by swapping the **RecordCounter** implementation with another that uses **GlideAggregate**.

Lab 3.3
Glide Aggregate

Reduce Database Load by Using GlideAggregate.

1. Navigate to **System Definition > Script Includes** and open **RecordCounter**.



2. Change the implementation of the **activeRecordsByField** method to leverage **GlideAggregate**.

```
var RecordCounter = Class.create();
RecordCounter.prototype = {
    initialize: function() {
    },

    activeRecordsByField: function(table, field, value) {
        var recordsByField = new GlideAggregate(table);
        recordsByField.groupBy(field);
        recordsByField.addAggregate("COUNT");
        recordsByField.addQuery(field, value + "");
        recordsByField.addActiveQuery();
        recordsByField.query();
        recordsByField.next();

        return recordsByField.getAggregate("COUNT");
    },

    type: 'RecordCounter'
};
```

Lab Success Verification

1. Open **INC0000039** and attempt to assign it to **Bow Ruggeri**. You should see **INC0000039** assigned to **someone else**.

2. All behavior from labs 3.1 and 3.2 should remain intact.