

# 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

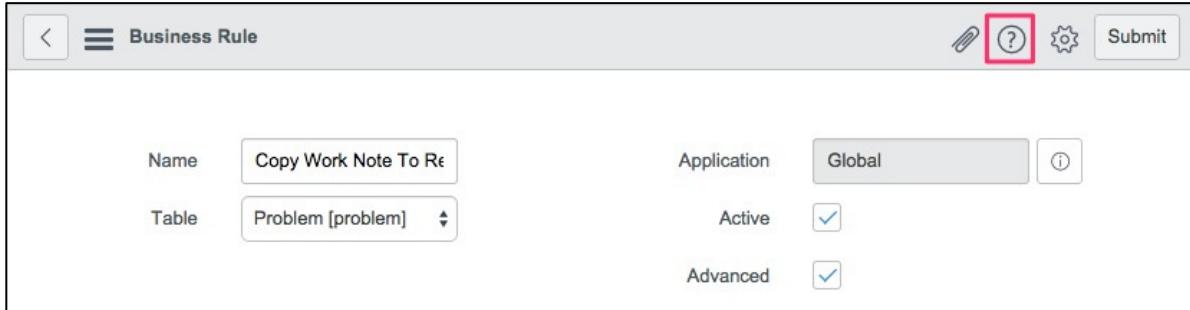
1. Navigate to **System Definition > Business Rules** and search by **Name** for **Copy Work Note To Related Incidents**.

The screenshot shows the ServiceNow System Administrator interface. At the top, it says "Welcome: System Administrator". Below that is a navigation bar with icons for Home, Business Rules, Metrics, System Definition, and Business Rules again. A red arrow points to the second "Business Rules" link in the System Definition section. The main area is titled "Business Rules" and shows a list of rules. The first rule is "Abort action if no license type" and the second is "Add Approver If Process Guide running". The "Advanced" tab is highlighted with a red border.

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



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



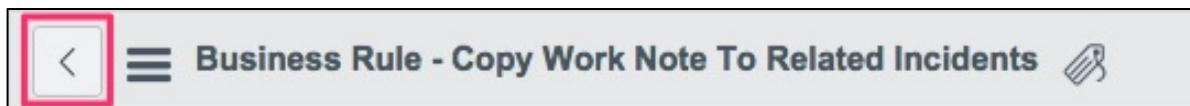
4. View the Business Rule **Script** and note that it uses a GlideRecord named “gr” and does an **update** operation on the incident table.

```
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.

The screenshot shows the ServiceNow search interface. At the top, there are buttons for 'Business Rules', 'New', 'Go to', and a search bar with the placeholder 'Name'. Below the search bar, a filter icon and the text 'All > Name = Active Change Requests' are visible. The search results table has columns for 'Name' (with a dropdown arrow), 'Search', and 'Search'. A row is selected, highlighted with a red box, showing the value '=Active Change Requests' in the 'Name' column. The table also includes a 'Details' icon, the name 'Active Change Requests', the table name 'Incident [incident]', and the value 'true' in the last column.

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

The screenshot shows the Business Rule configuration interface. At the top, there are tabs for 'When to run', 'Actions', and 'Advanced'. The 'Advanced' tab is highlighted with a red box.

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

The screenshot shows the Business Rule configuration interface under the 'Advanced' tab. It displays the 'Name' field set to 'Active Change Requests' and the 'Table' field set to 'Incident [incident]' (highlighted with a red box). Below these fields, there are tabs for 'When to run', 'Actions', and 'Advanced'. Under the 'Advanced' tab, there is a 'Condition' field containing an empty box. In the 'Script' section, there is a toolbar with various icons and a code editor containing the following JavaScript:

```
1 var gr = new GlideRecord("change_request");
2 gr.addQuery("parent", current.sys_id);
3 gr.query();
4
5 gs.addInfoMessage(current.number + " has " + gr.get
```

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**.

The screenshot shows the ServiceNow Problem search interface. On the left, a sidebar menu titled "Problem" lists several options: Create New, Assigned to me, Known Errors, Open, Pending, All (which has a red arrow pointing to it), and Overview. The main area displays a list of problems with columns for Number, Short description, and a small icon. The problem PRB0000050 is highlighted with a red box.

Number	Short description
PRB0000055	Email down
PRB0000053	Slow switching
PRB0000051	Exchange server outage
PRB0000050	Switch occasionally drops connections

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

The screenshot shows the ServiceNow Problem detail view for PRB0000050. At the top, there's a header with the title "Problem - PRB0000050". Below it, the problem details are listed: Description (Switch occasionally drops connections), Impact (3 - Low), Opened by (Empty), and Priority (4 - Low). There are "Update" and "Delete" buttons at the bottom of this section. The "Related Links" section includes links to Communicate Workaround, Post Knowledge, and Post News. The "Incidents" tab is selected, and the "Edit..." button is highlighted with a red box. Below the tabs, there's another search bar and a list of incidents related to the problem.

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

Edit Members

Add Filter Run filter ?

-- choose field -- -- oper -- -- value --

Collection

Incidents List

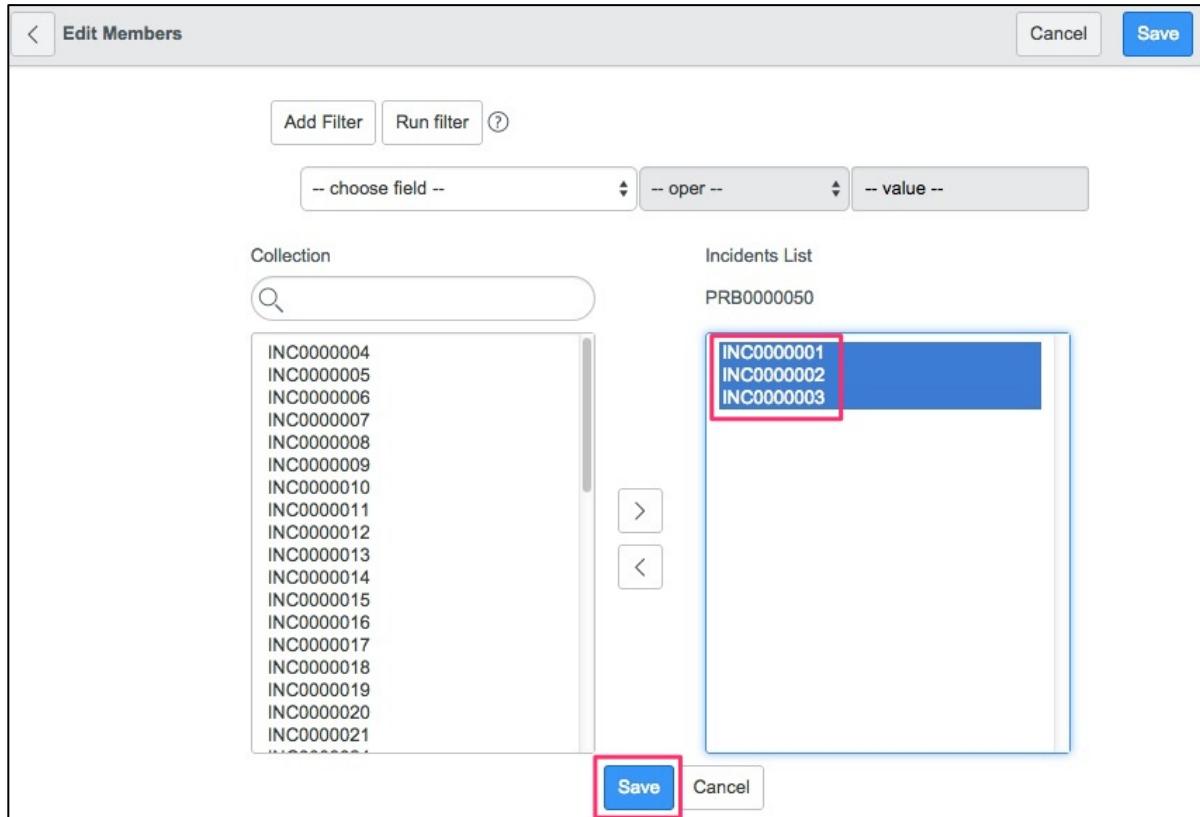
PRB0000050

INC0000004  
INC0000005  
INC0000006  
INC0000007  
INC0000008  
INC0000009  
INC0000010  
INC0000011  
INC0000012  
INC0000013  
INC0000014  
INC0000015  
INC0000016  
INC0000017  
INC0000018  
INC0000019  
INC0000020  
INC0000021

> <

INC0000001  
INC0000002  
INC0000003

Save Cancel



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

Problem - PRB0000050

Workaround

Work notes Add a work note here!



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.

The screenshot shows a ServiceNow interface for a 'Problem' record with number PRB0000050. The title bar says 'Problem - PRB0000050'. The main content area has a blue header 'Copying Work Note to INC0000001' followed by the message 'INC0000001 has 3 Change Requests associated with it.' Below this, three lines of text are listed: 'Copying Work Note to CHG0000010', 'Copying Work Note to CHG0000007', and 'Copying Work Note to CHG0000004'. The last three lines are highlighted with a red rectangular box. At the bottom, there are two input fields: 'Number' containing 'PRB0000050' and 'Configuration item' containing 'ny8500-nbxs08'. To the right of the configuration item field are three small icons: a magnifying glass, a gear, and an information circle.

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.

```
1 (function() {
2     var gr = new GlideRecord("incident");
3     gr.addQuery("problem_id", current.sys_id);
4     gr.query();
5
6     while(gr.next()) {
7         gs.addInfoMessage("Copying Work Note to " + gr.getDisplayValue());
8         gr.comments = current.work_notes;
9         gr.update();
10    }
11 })();
```

16. Save the **Business Rule**.

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

```
1 (function() {
2     var gr = new GlideRecord("change_request");
3     gr.addQuery("parent", current.sys_id);
4     gr.query();
5
6     gs.addInfoMessage(current.number + " has " + gr.getRowCount() + " Change Requests associated with it.");
7 })();
```

18. Save the **Business Rule**.

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

Work notes

Things should work out this time...

20. Notice the result this time is as expected.

The screenshot shows a list of work notes being copied to three different records. The header of the list indicates "Copying Work Note to INC0000001". Below this, it says "INC0000001 has 3 Change Requests associated with it." The next item in the list is "Copying Work Note to INC0000002", followed by the message "INC0000002 has 0 Change Requests associated with it.". The final item in the list is "Copying Work Note to INC0000003", followed by the message "INC0000003 has 0 Change Requests associated with it.".

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**.

The screenshot shows the ServiceNow interface for managing business rules. The top navigation bar says "Welcome: System Administrator". On the left, there's a sidebar with "Metrics" and "System Definition" sections. Under "System Definition", the "Business Rules" item is highlighted with a red box. The main content area has a title bar with "Business Rules", a "New" button, and a search bar. Below that, a filter bar shows "All > Name contains Count Change". The main table lists one row: "\*Count Change" under "Name", with a "Search" button next to it. At the bottom of the table, there's a link "Count Change Tasks By Assignee" which is also highlighted with a red box. To the right of the table, there's a note "Change Request [change\_request]".

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.

The screenshot shows the configuration of a business rule named "Count Change Tasks By Assignee". The "Table" is set to "Change Request [change\_request]". The "When to run" section is highlighted with a red box, showing "When" set to "display" and "Order" set to 100. Other tabs like "Actions" and "Advanced" are visible but not selected.

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

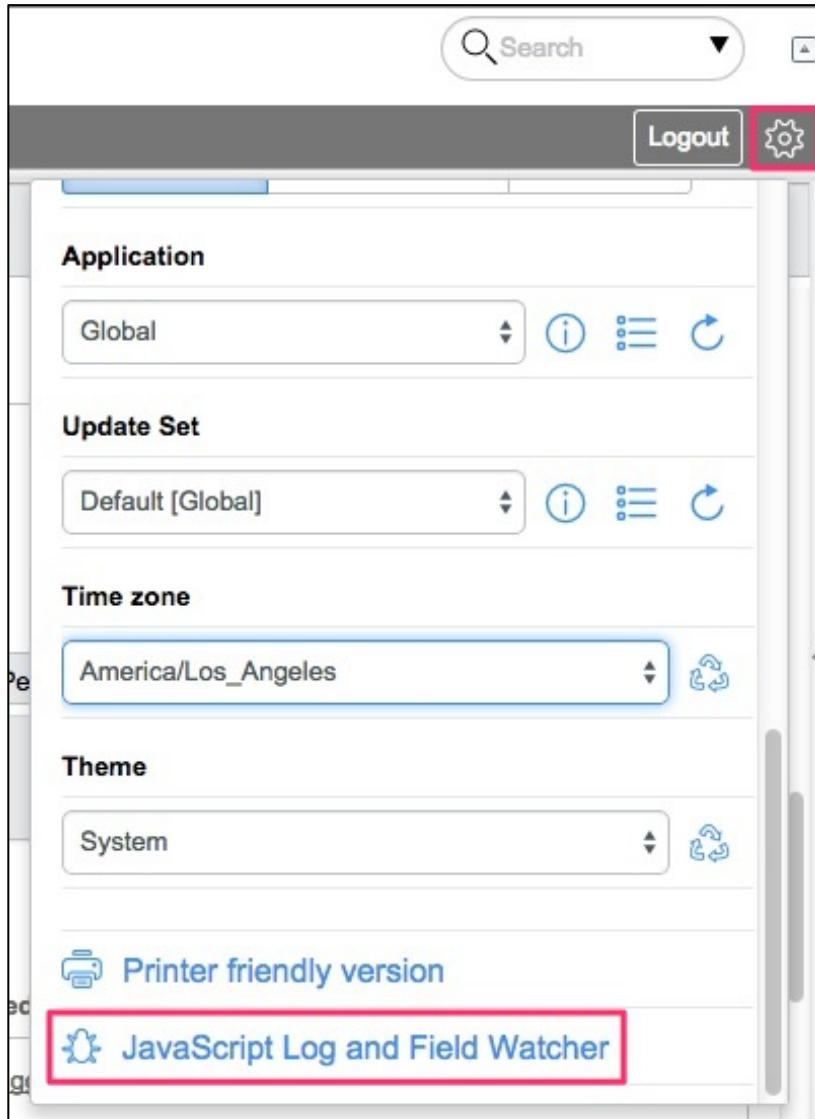
The screenshot shows the "Change Requests" list view. The sidebar on the left has a "Change" menu with "All" selected. The main area lists three change requests: CHG0000015 (Unix update), CHG0000014 (CMS App FLX), and CHG0000013 (Oracle FLX). The row for CHG0000013 is highlighted with a red box.

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.

The screenshot shows the ServiceNow interface for a Change Request (CHG0000013). At the top, two messages are displayed: "Don Goodliffe has 4 Change Tasks assigned." and "Fred Luddy has 1 Change Tasks assigned." Below this, the Change Request details are shown, including the number CHG0000013. A navigation bar at the top of the main content area includes tabs for Affected Cls, Impacted Services/Cls, Approvers, Change Tasks (3), Problems, and Incidents Pending Cha. The Change Tasks tab is selected, showing a list of three tasks. The "Assigned to" column for these tasks is highlighted with a red box, listing Bow Ruggeri, Beth Anglin, and Don Goodliffe.

Assigned to
Bow Ruggeri
Beth Anglin
Don Goodliffe

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.

The screenshot shows a sidebar with 'Conflict Properties', 'System Localization' (selected), 'Exchange Rates', 'Load Exchange Rates', and 'System Logs'. Below the sidebar is a table with three rows:

<input type="checkbox"/>	<i>(i)</i>	<a href="#">CHG0000011</a>	Another Java Application Server change
<input type="checkbox"/>	<i>(i)</i>	<a href="#">CHG0000010</a>	Java Application Server change
<input type="checkbox"/>	<i>(i)</i>	<a href="#">CHG0000009</a>	Apply patches 10.2.0.1 to 10.2.0.3

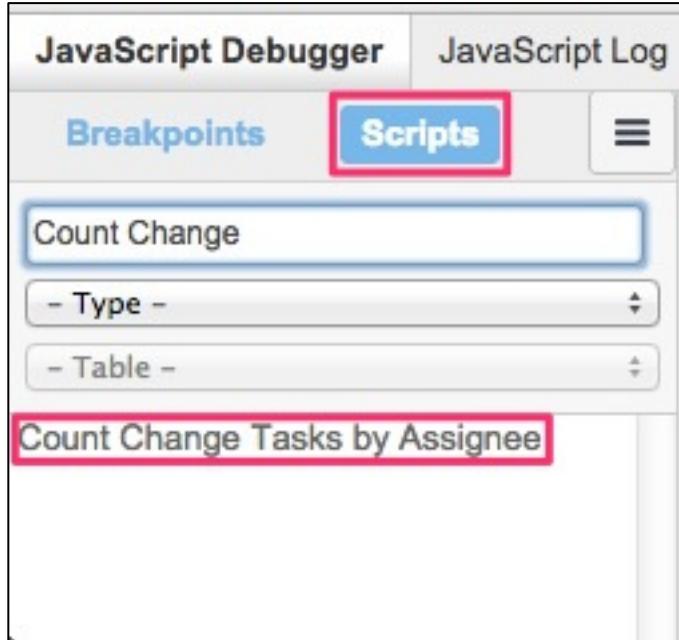
Below the table are three tabs: 'JavaScript Debugger' (highlighted with a red box), 'JavaScript Log', and 'Field Watcher'. The 'JavaScript Log' tab contains the following log entries:

```
21:11:43 (515) jsdebug.do running inline scripts, count: 0
21:11:43 (519) jsdebug.do runBeforeRender
21:11:43 (521) jsdebug.do runAfterAllLoaded, functions: 6
21:11:43 (523) jsdebug.do fireAllChangeHandlers start
21:11:43 (524) jsdebug.do fireAllChangeHandlers end
21:11:43 (524) jsdebug.do late load functions: 2
21:11:43 (530) jsdebug.do [00:00:00.008] runAfterAllLoaded finished
21:11:43 (771) jsdebug.do after page loaded starting
```

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

The screenshot shows the 'JavaScript Debugger' interface with the 'Breakpoints' tab selected. The interface includes a toolbar with icons for power, play, stop, and navigation, and a message 'No Breakpoints'.

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



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

The screenshot shows the ServiceNow JavaScript Debugger interface with the 'Scripts' tab selected. On the left, the search results for 'Count Change Tasks by Assignee' are shown. On the right, the script code is displayed in a code editor:

```
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") + " has " + otherTasksForUser.getValue("COUNT"));
21     }
22 }
```

At the bottom right of the code editor, there are status indicators: 'Table: sys\_script' and 'ID: d3d6fb41250331007f44'.

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

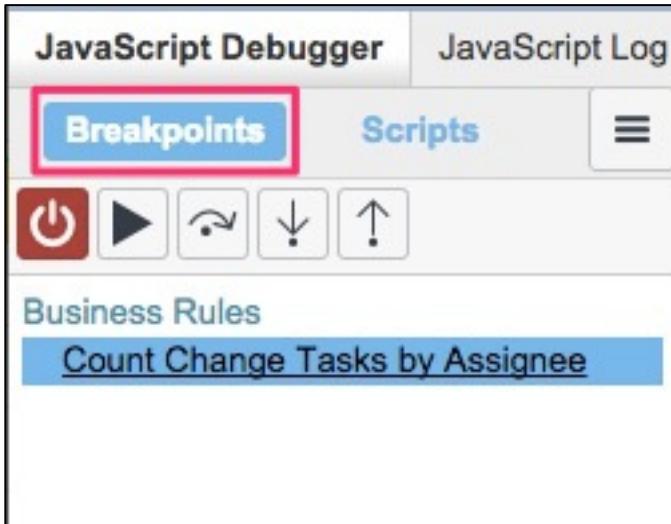
11. View **CHG0000013** and notice that execution has paused on the second line. A red breakpoint indicates that execution has paused on the breakpoint.

The screenshot shows the ServiceNow Service Automation interface. On the left, there's a sidebar with various navigation links like 'Change', 'Create New', 'Open', 'Closed', etc. The main area displays a list of 'Change Requests' with columns for 'Number', 'Short description', 'Type', 'State', and 'Assigned to'. One specific change request, CHG0000013, is highlighted with a red border. Below this, a 'JavaScript Debugger' window is open. It has tabs for 'Breakpoints' (selected), 'Scripts', and 'Field Watcher'. The 'Scripts' tab shows a script with several lines of code. Line 4 is highlighted with a red border and contains the breakpoint information: 'var changeTaskAssignees = [current.assigned\_to + ""]';. To the right of the debugger, there are sections for 'Call Stack' and 'Variables'. The 'Variables' section is highlighted with a blue box. It shows a 'Watch Variables' section with a 'Local' subsection. The local variables listed are: changeTaskAssignees: undefined, changeTasks: undefined, current: GlideRecord, g\_scratchpad: {}, and otherTasksForUser: undefined.

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.

This screenshot shows a detailed view of the ServiceNow JavaScript Debugger's 'Variables' section. The 'Variables' tab is selected and highlighted with a blue box. Below it, the 'Local' section is also highlighted with a red box. The local variables listed are: changeTaskAssignees: undefined, changeTasks: undefined, current: GlideRecord, g\_scratchpad: {}, and otherTasksForUser: undefined. Other sections visible include 'Call Stack' and 'Closures'.

13. Click the **Breakpoints** button.

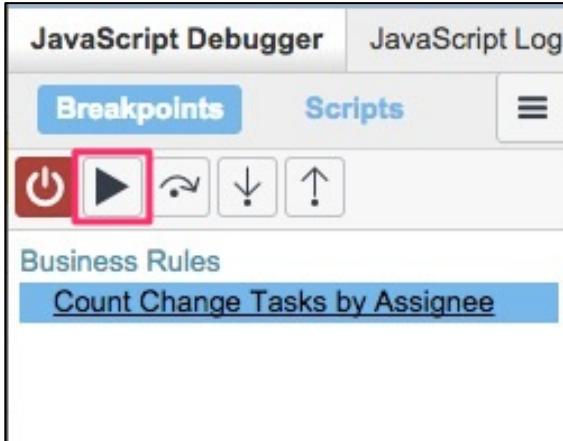


14. Add a breakpoint on **line 10**.

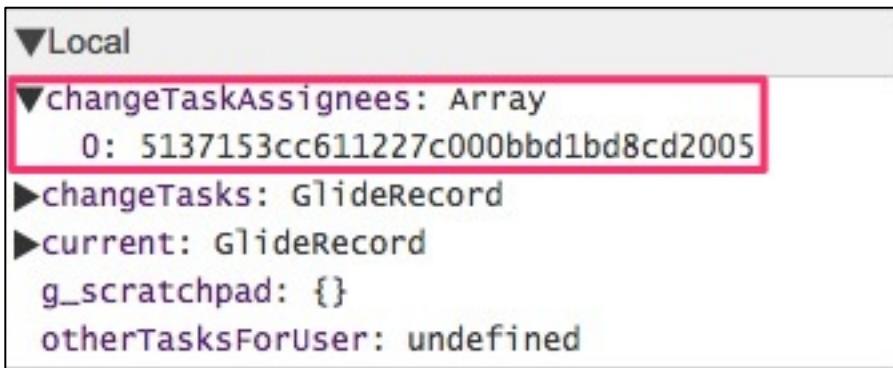
A screenshot of the ServiceNow JavaScript Debugger. The "Breakpoints" tab is selected. In the code editor on the right, line 10 is highlighted with a red box. The code is as follows:

```
1 function onDisplay(current, g_scratchpad) {  
2     //This function will be automatically called when this rule is p  
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);
```

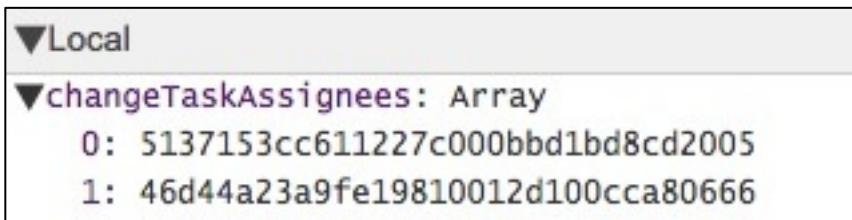
15. Press the **Play** button to advance to the next breakpoint.



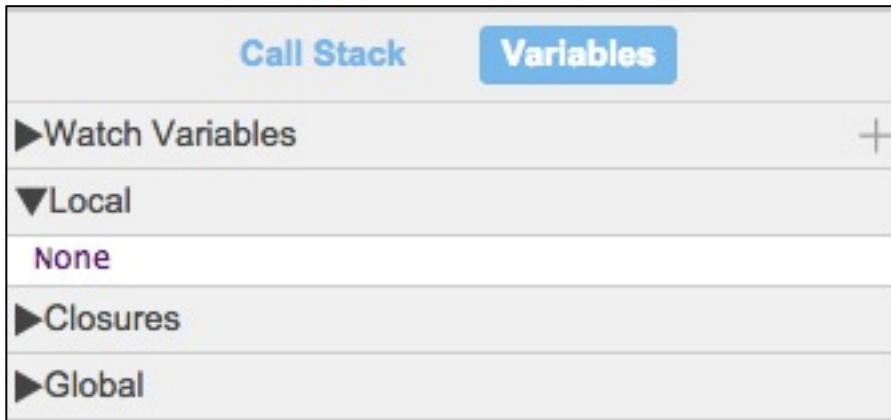
16. Expand the **changeTaskAssignees** variable to see what values it contains.



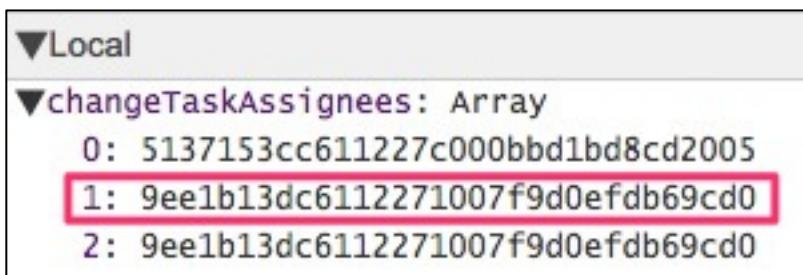
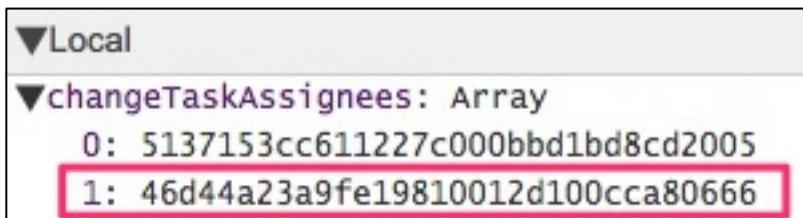
17. Click the **Play** button to continue execution. The debugger pauses on **line 10** again because it is iterating over a GlideRecord collection with multiple entries. It pauses on **line 10** once per change task. Notice that values in the **changeTaskAssignees** are updated as the script iterates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

```
cord("change_task");
request", current.sys_id);

changeTasks.assigned_to + "");
```

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.

JavaScript Debugger	JavaScript Log	Field Watcher
<b>Breakpoints</b>	<b>Scripts</b>	
		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     }

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.

The screenshot shows a ServiceNow interface for a 'Change Request - CHG0000013'. At the top, there's a back arrow, a menu icon, the title 'Change Request - CHG0000013', and a clipboard icon. Below the title, four messages are listed in blue boxes:

- Beth Anglin has 1 Change Tasks assigned.
- Bow Ruggeri has 5 Change Tasks assigned.
- Don Goodliffe has 4 Change Tasks assigned.
- Fred Luddy has 1 Change Tasks assigned.

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

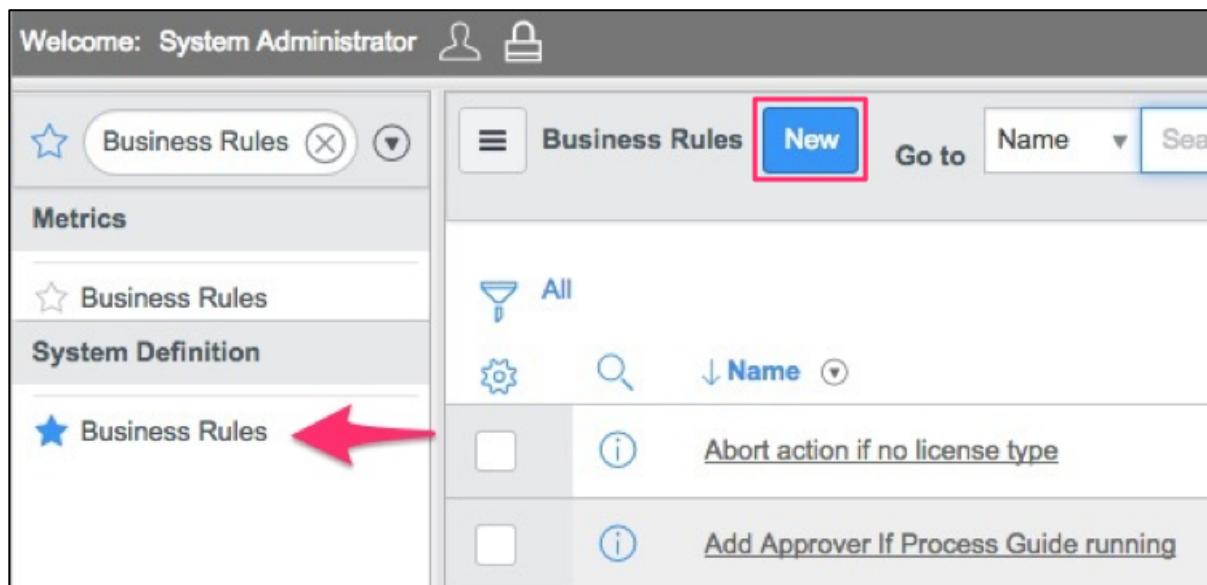
## Lab Goal

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

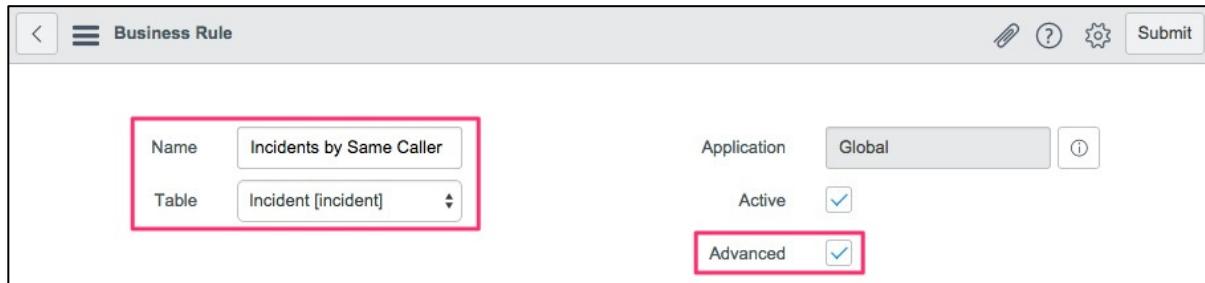
**Lab 3.1**  
**Do Not**  
**Repeat**  
**Yourself**

### Consolidating Common Code Using Script Includes

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



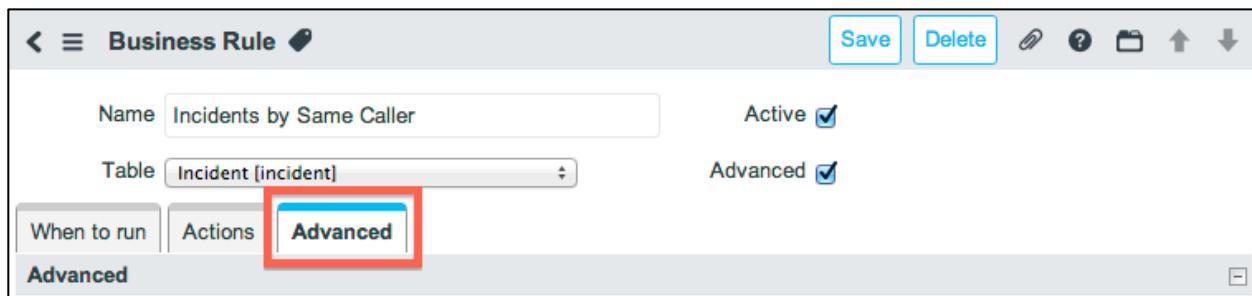
2. Create a new **Advanced Rule** called **Incidents by Same Caller** and configure it to run against **Incident**.



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



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

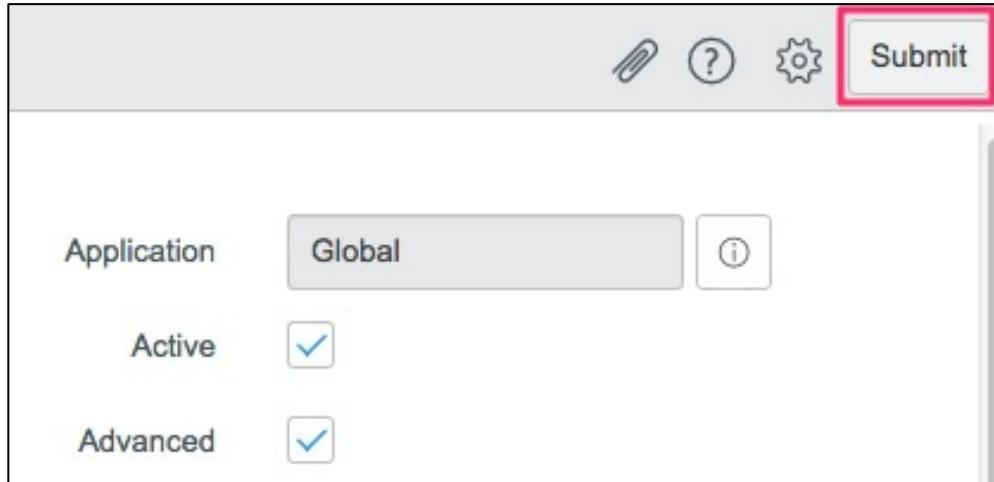


5. Write a script for **Incidents by Same Caller** that queries Incident for records where **caller\_id** equals **current.caller\_id**. Use **gs.addInfoMessage** to display the result on screen.

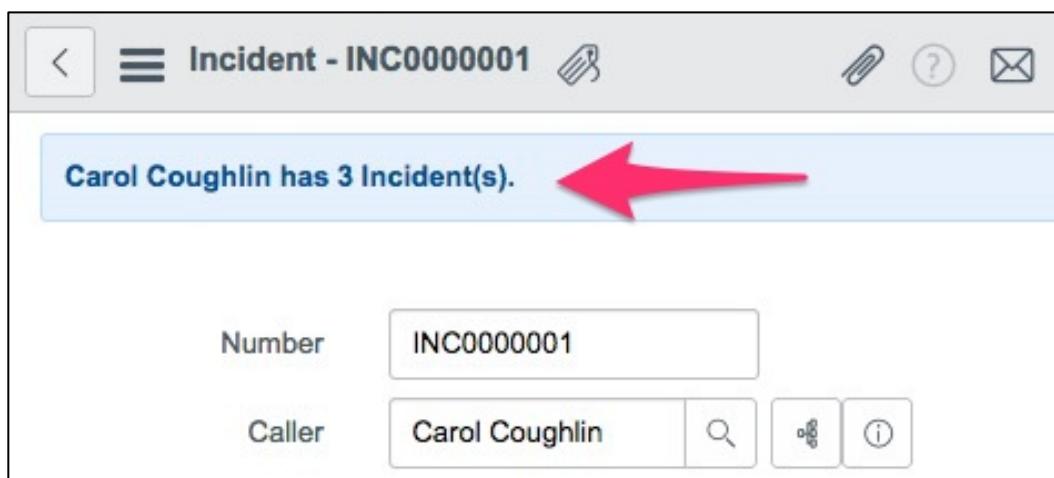
```
function onDisplay(current, g_scratchpad) {
    var incidentsBySameCaller = new GlideRecord("incident");
    incidentsBySameCaller.addQuery("caller_id", current.caller_id);
    incidentsBySameCaller.query();

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

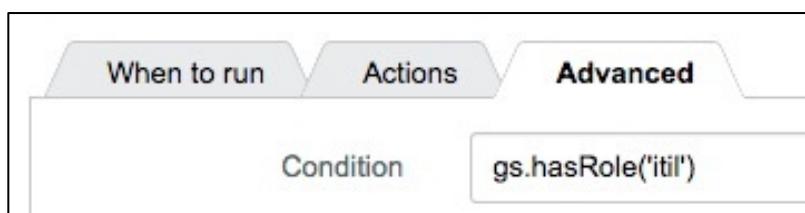
6. Save Incidents by **Same Caller** by clicking **Submit**.



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.

The screenshot shows the ServiceNow Service Automation interface. In the top navigation bar, the title is "servicenow™ Service Automation" and the welcome message is "Welcome: System Administrator". A user icon and a lock icon are also present. On the left sidebar, there are two items: "Toggle Navigator" (ctrl + opt + n) and "List and Form View" (ctrl + opt + v). Below the sidebar, there is a search bar with a star icon, a "Filter" button, and a dropdown arrow. To the right of the search bar, there is a "Incidents [Self Service view]" button and a "New" button. A red box highlights the user icon in the top bar. A red box also highlights the search input field in the "Impersonate User" dialog box. The "Impersonate User" dialog box has a title "Impersonate User" and a close button. It contains a section titled "Recent Impersonations" with a list: "System Administrator" and "Joe Employee". Below this is a search input field containing "Joe Employee", a magnifying glass icon, and an information icon. At the bottom of the dialog are "Cancel" and "OK" buttons, with the "OK" button highlighted by a red box.

10. Open one of Joe Employee's incidents and confirm that no incident counts are displayed.

The screenshot shows the ServiceNow navigation bar with the title "Impersonating: Joe Employee". Below it is a "Self-Service" menu with several items: "Live Feed", "Homepage", "Service Catalog", "Knowledge", "Help the Help Desk", "My Task Boards", and "Incidents". A large red arrow points to the "Incidents" link. Below the menu is a detailed view of an incident record for "INC0000051" with fields for "Number" and "Caller".

Number	INC0000051
Caller	Joe Employee

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.

The screenshot shows the "Business Rule - Incidents by Same Caller" page. A context menu is open over the "Name" field, with "Insert and Stay" highlighted. A red arrow points to this option. A tooltip explains that "Insert and Stay" creates a copy of the current record with any modified values, and redirects to the new record.

Name	Incidents by Same Caller
Table	Incident [incident]

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



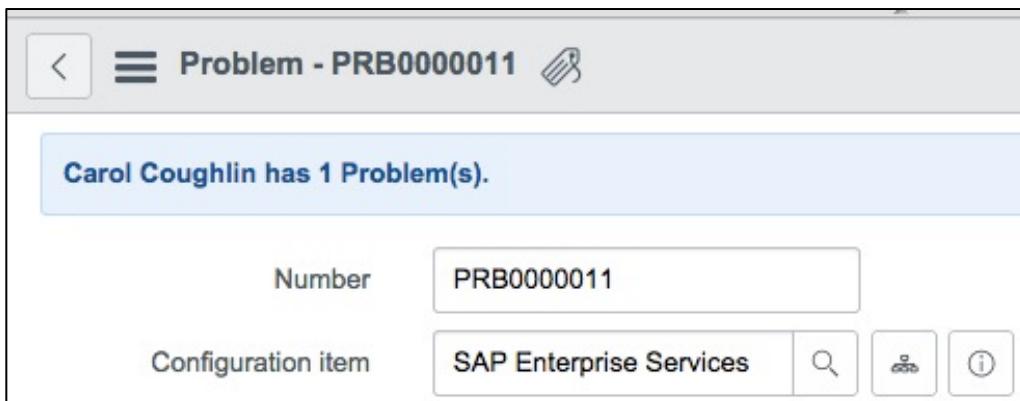
14. Change the script to query **Problem**. The field being filtered is **opened\_by**.

```
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:**

Condition	gs.hasRole('itil') && current.caller_id.company.u_critical_account
-----------	--

**Problem:**

Condition	gs.hasRole('itil') && current.opened_by.company.u_critical_account
-----------	--

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

The screenshot shows a ServiceNow problem detail view. At the top, there is a header with a back arrow, a menu icon, the title "Problem - PRB0000011", and a edit icon. Below the header, there are two main sections: "Number" and "Configuration item". The "Number" section contains a text input field with the value "PRB0000011". The "Configuration item" section contains a text input field with the value "SAP Enterprise Services", a search icon, and two small circular icons. The entire interface is contained within a white rectangular box with a thin black border.

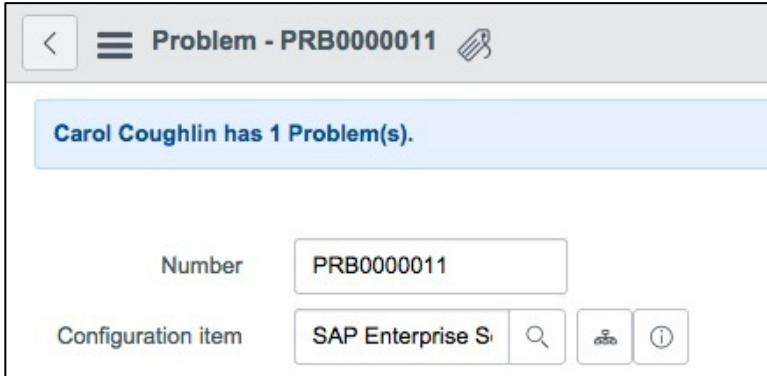
19. Open the **Company** record for ACME Americas

The screenshot shows the 'Companies' list view in ServiceNow. On the left, there's a sidebar with 'Organization' and 'User Administration' sections, each containing a 'Companies' item. A large red arrow points to the 'Companies' item under 'Organization'. The main panel displays a list of companies with columns for selection, information icon, and name. The names listed are 3Com, Acer, ACME Africa, and ACME Americas. The 'ACME Americas' row is highlighted with a red box.

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

The screenshot shows the 'Company - ACME Americas [Customer view]' edit screen. The top bar includes back, list, edit, help, gear, and 'Update' (which is highlighted with a red box) buttons. The form contains fields for Name (ACME Americas), Phone, Fax phone, Customer (with a checked checkbox), Stock symbol, Stock price, Street, City, State / Province, Zip / Postal code, and Critical Account (which is also checked and highlighted with a red box). The 'Update' button is again highlighted with a red box at the bottom right of the form area.

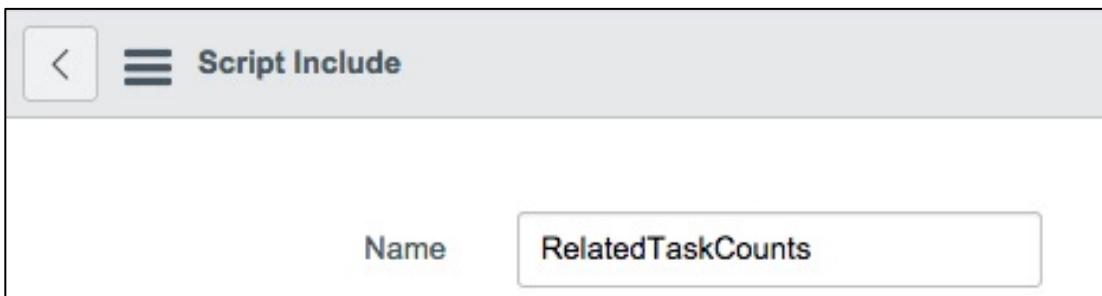
21. Open **PRB0000011** and confirm that PRB counts for **Carol Coughlin** are visible.



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:

The screenshot shows the ServiceNow interface for an incident record. At the top, it says "Incident - INC0000050". Below that, a message states "Jerrod Bennett has 1 active Incident(s)". The main details section shows the "Number" as "INC0000050" and the "Caller" as "Jerrod Bennett". There are also small icons for "VIP" and search functions. On the right side of the details section, there are three small buttons: a magnifying glass, a gear, and an information icon.

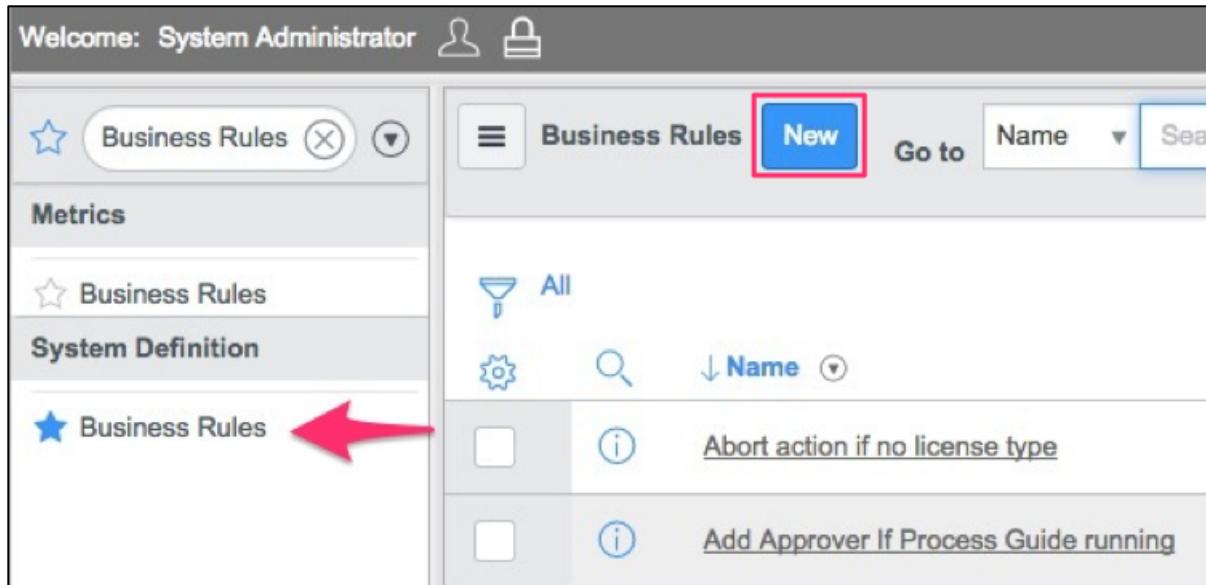
## Lab Goal

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**

The screenshot shows the 'Business Rule' configuration interface. It includes fields for 'Name' (set to 'Fair Incident Assignment'), 'Table' (set to 'Incident [incident]'), 'Application' (set to 'Global'), 'Active' (checked), and 'Advanced' (checked). The 'Advanced' checkbox is highlighted with a red box.

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

The screenshot shows the 'When to run' tab of the Business Rule configuration. It has three tabs: 'When to run', 'Actions', and 'Advanced'. Under 'When to run', 'When' is set to 'before' and 'Order' is set to '100'. Under 'Actions', 'Insert' and 'Update' are checked, while 'Delete' and 'Query' are unchecked. The 'Insert' and 'Update' checkboxes are highlighted with a red box.

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

The screenshot shows the 'Condition' field in the Business Rule configuration. The condition is defined as: `previous.assigned_to.nil() && !current.assigned_to.nil()`.

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();
    }

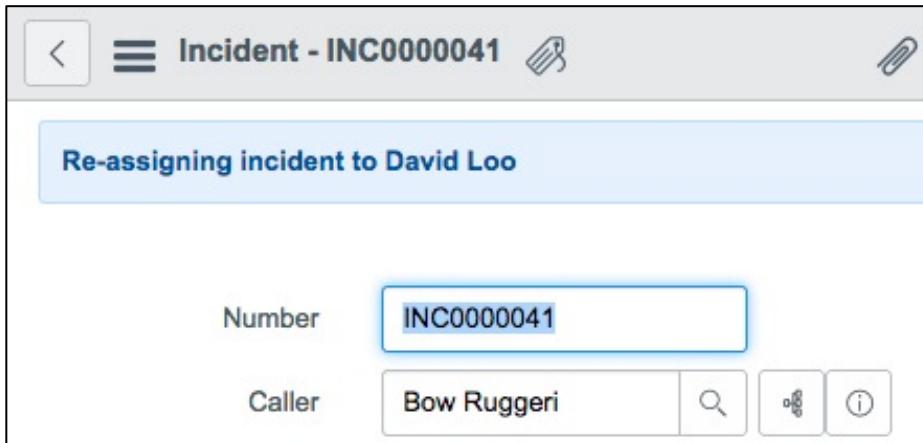
    function findUserWithLessThanThree() {
        var usersInThisGroup = new GlideRecord("sys_user_grmember");
        usersInThisGroup.addQuery("group", current.assignment_group);
        usersInThisGroup.query();

        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 illustrates 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**.

The screenshot shows the ServiceNow System Definition > Script Includes page. The left sidebar has categories like Metrics, MID Server, and System Definition. The main area shows a list of script includes. A filter bar at the top says "All > Name = RecordCounter". The list table has columns for Name, Application, and Active. One row is selected, showing "RecordCounter" in the Name column, "Global" in the Application column, and "true" in the Active column.

Name	Application	Active
RecordCounter	Global	true

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.