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

Lab Guide

Scripting 201:

Client Scripting for ServiceNow

Matthias Martini & Oliver Schmitt

Login / Passwords:

admin / Knowledge15

itil / Knowledge15

employee / Knowledge15

This

Page

Intentionally

Left

Blank

Lab Goal

This lab explains how to create a Client Script that automatically adds the callers manager to the watch list when the incident is a Priority 1.

Lab 1.1 Auto Watch List for Priority 1 Incidents

Auto Watch List for Priority 1 Incidents

1. Log in to the instance URL on the cover page using the provided admin credentials.

2. Open the **Script Include** list: **System Definition > Script Includes**.

3. Create a new **Script Include**.

4. Configure the **Script Include trigger**.

   Name: **UserUtilsAjax**
   Active: **Selected** (checked)
   Client Callable: **Selected** (checked)
   Description: **Helper class for user related Ajax calls**

5. Here is the pseudo code for the script:

   Create a new class called UserUtilsAjax
   Create an object from the new class with properties inheritable by other objects and which extends the AbstractAjaxProcessor class.
   Add a method to the new object called getManager
   Retrieve the user record using the correct input parameter and return the matching manager's SysID

6. Write the **Script Include**:

```
var UserUtilsAjax = Class.create();
UserUtilsAjax.prototype = Object.extendsObject(AbstractAjaxProcesor, {
    getManager: function(){
        var userID = this.getParameter('sysparm_userID');
        var user = new GlideRecord('sys_user');
        if(user.get(userID)){
            return user.manager;
        }else{
            return -1;
            }
    },
    type: 'UserUtilsAjax'
});
```

7. **Submit** the **Script Include**.

8. Open the Client script list: **System Definition > Client Scripts**.

9. Create a new **Client script**.

10. Configure the **Client script trigger**.

    Name: **AutoWatchlistPrio1**
    Active: **Selected** (checked)
    Global: **Selected** (checked)
    Type: **onChange**
    Table: **Incident** [incident]
    Field name: **Priority**
    Inherited: **Not selected** (unchecked)
    Description: **rege**

11. Here is the pseudo code for the script:

    Create an instance of the GlideAjax object called UserUtilsAjax
    Add a param to call the getManager method
    Add a param to pass the callers SysID
    Use the getXML method and the callback function GetManagerParse to execute the Script Include

    Pass the response returned from the Script Include to the callback function
    Locate the answer variable in the returned XML and store the value in a variable
    Add the retrieved SysID to the Watchlist

12. Write the script:

```
function onChange(control,oldValue, newValue, isLoading, isTemplate) {
    if (isLoading || newValue == '') {
        return;
    }
    //dont continue if new priority is not 1 or there is no caller provided
    if(newValue !=1 || g_form.getValue('caller_id') == '') {
        return;
    }

    var ga = new GlideAjax('UserUtilsAjax');
    ga.addParam('sysparm_name', 'getManager');
    ga.addParam('sysparm_userID', g_form.getValue('caller_id'));
    ga.getXML(getManagerParse);
}

function getManagerParse(response){
 var managerID = response.responseXML.documentElement.getAttribute('answer');
    if (managerID == -1 || managerID == ''){
        jslog('Error retrieving callers manager');
    }else{
        g_form.setValue('watch_list', managerID);
        g_form.addInfoMessage("Caller's manager added to watchlist due to critical priority");
    }
}
```

13. **Save** the Client script.

14. Create a **new** Incident: **Incident > Create New**.

15. Fill in the following fields:

    Caller: **Beth Anglin**
    Impact: **1 - High**
    Urgency: **1 - High**

16. The Priority is automatically calculated as **1 – Critical**.

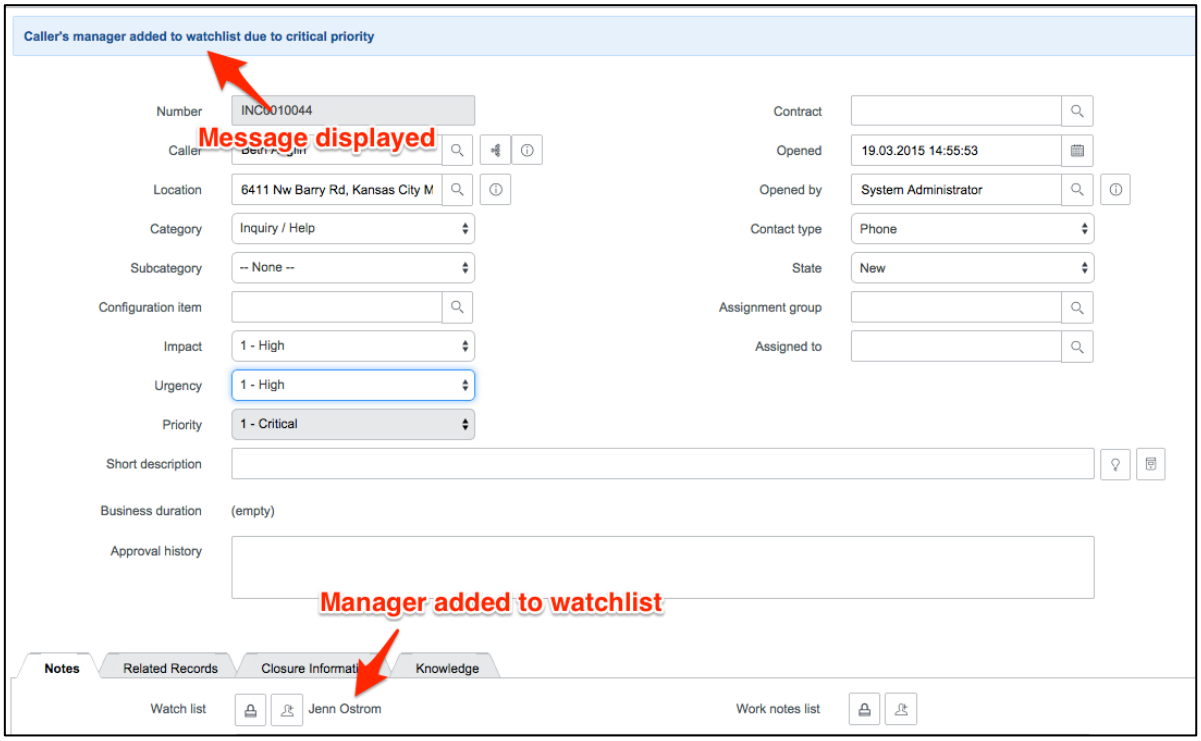
Challenge:

When setting the value of a reference or list field using **g_form.setValue()** and only passing a **sys_id** as parameter, this triggers an additional server roundtrip to get the matching display value.

Adjust your script include and client script so that this roundtrip is no longer necessary.

Lab Success Verification

1. The screen should look like the example.



Lab Goal

This lab explains how to create new Context Menu entries to simplify opening lists and record in new browser tabs.

Lab 2.1 "Open in New Tab" Context Menu

"Open in new tab" context menu

1. Open the Context menu list: **System UI > UI Context Menus**.

2. Create a **new** Context menu.

3. Configure the Context menu trigger.

   Table: **Global**
   Menu: **List title**
   Type: **Action**
   Name: **Open in new tab**

4. Write the script:

```
window.open(g_list.getReferringURL());
```

5. **Save** the Context menu.

6. Create a **new** Context menu.

7. Configure the Context menu trigger.

   Table: **Global**
   Menu: **List row**
   Type: **Action**
   Name: **Open in new tab**

8. Write the script:

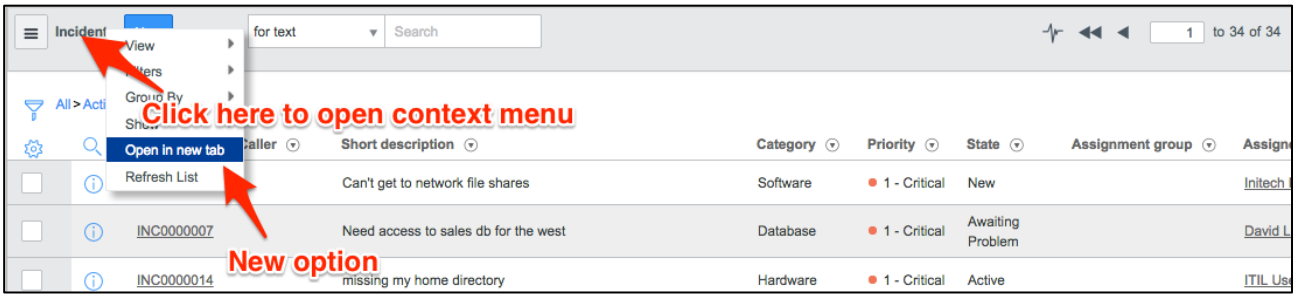
```
window.open('/' + g_list.getListName() + '.do?sys_id=' + g_sysId);
```

9. **Save** the Context menu.

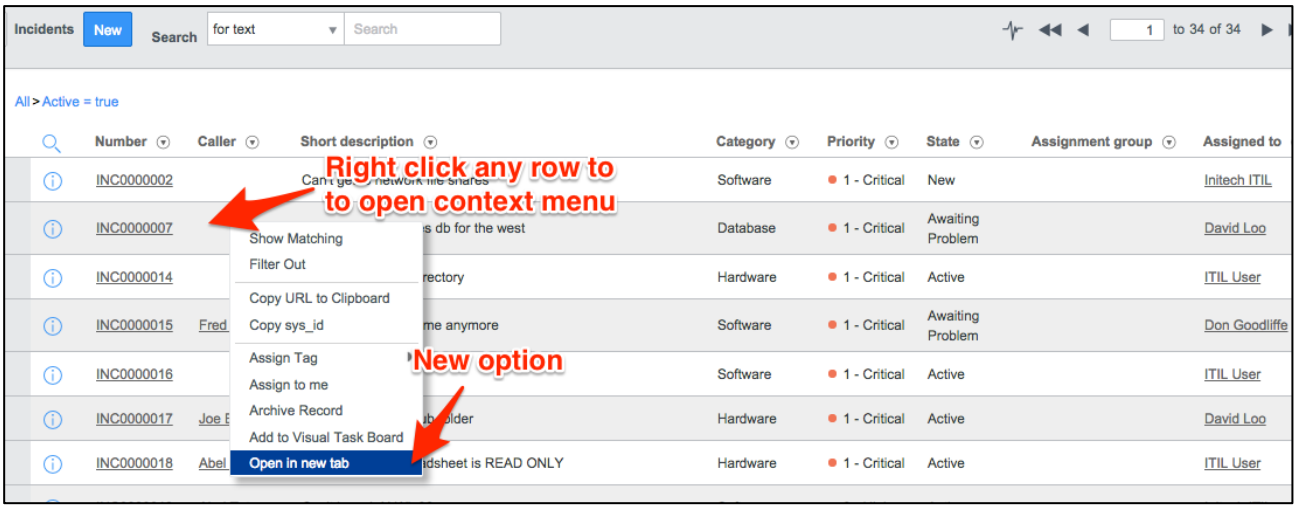
Lab Success Verification

1. When looking at any record list, the **Open in new tab** context menu should now be available for the list and every record. Select the menu to open the list/record in a new tab.

Open list in new tab



Open record in new tab



Lab Goal

This lab explains how to create a new UI Action that lets you create a new Configuration Item without disturbing the Incident workflow.

Lab 3.1 Create Missing CIs from Incident

Create Missing CIs from Incident

1. Open the UI Action list: **System Definition > UI Actions**.

2. Create a **New** UI Action.

3. Configure the UI Action trigger.

   Name: **Create CI**
   Table: **Incident** [incident]
   Order: **100**
   Action name: **incident_new_ci**
   Active: **Selected** (checked)
   Show insert: **Selected** (checked)
   Show update: **Selected** (checked)
   Client: Selected (**checked**)
   Form button: **Selected** (checked)
   Hint: **Create a new CI to log the incident against**

4. Define the **onClick** property of the created UI Action.

```
onClick: ciPopup();
```

5. Add a condition to only show the UI Action in case no Configuration Item has yet been selected.

   Condition: **current.cmdb_ci.nil()**

6. Write the script:

```
function ciPopup() {
    var dialog = new GlideDialogForm('Create Server CI', 'cmdb_ci_server', populateCmdbCIField);
    dialog.setTitle('Create Server CI');
    dialog.addParm('sysparm_view', 'default');
    dialog.addParm('sysparm_form_only', 'true');
    dialog.render();
}

function populateCmdbCIField(action, sys_id, table, displayValue) {
    g_form.setValue('cmdb_ci', sys_id);
}
```

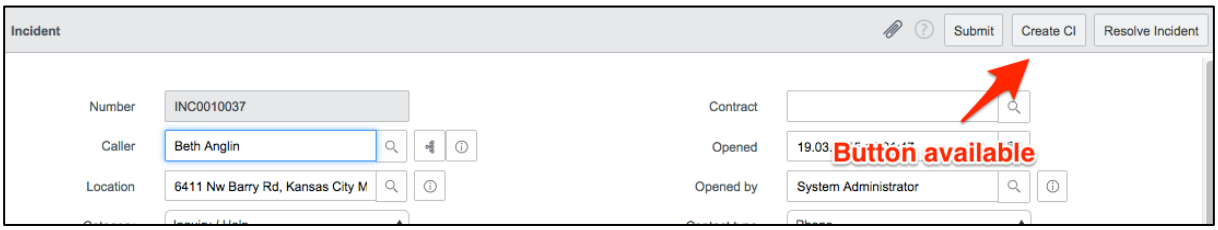
7. **Save** the UI Action.

Challenge

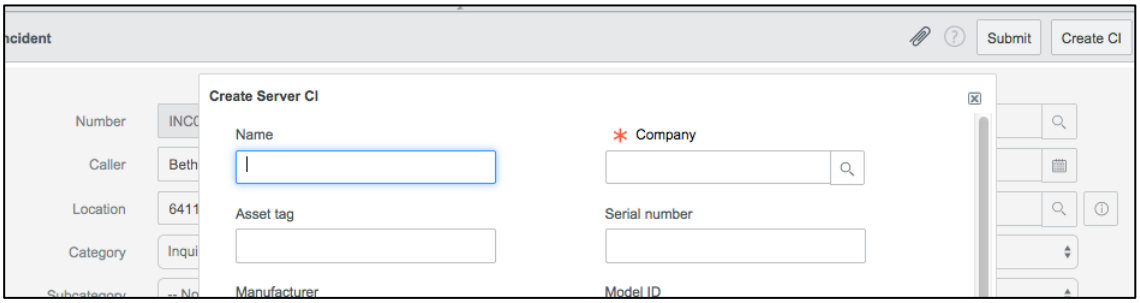
Adjust the UI Action so it is only available if the user has **write** access to the current record and also has the role **itil**.

Lab Success Verification

1. When you create a new Incident, you should now see the new UI Action.



2. When you click the button, a new dialog window opens showing a blank CI server form.



3. When you fill in all server information and submit the form, a new server record is created in the CMDB and the new CI is automatically used in the current Incident.

4. After you submit the Incident, the button is no longer visible due to the added condition that checks whether the Configuration Item field is empty.

Lab Goal

This lab explains how to create a UI Policy that makes some fields mandatory and raises the Priority in order to put the Service Desk's focus on the most critical changes planned.

Lab 4.1 Focus on Most Important Changes

Focus on Most Important Changes

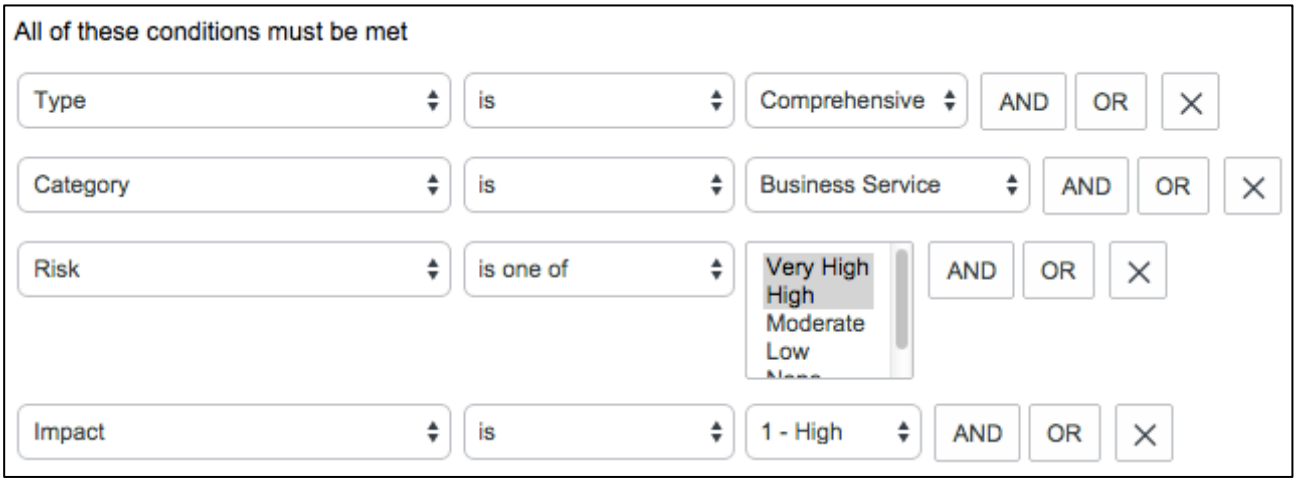
1. Open the UI Policy list: **System UI > UI Policies**.

2. Create a **new** UI Policy.

3. Configure the UI Policy trigger.

   Table: **Change Request [change_request]**
   Reverse if false: **Selected (checked)**
   Order: **100**
   On load: **Selected** (checked)
   Run scripts: **Selected** (checked)
   Run scripts in UI type: **Desktop**
   Active: **Selected** (checked)
   Inherit: **Not selected** (unchecked)
   Short description: **Focus on most important changes**

4. Define the condition for when to apply the UI Policy:



5. Open the **Script** tab and fill in the **Execute if true** script:

```
function onCondition() {
    g_form.setValue('priority', 1);
    g_form.addInfoMessage('This is a critical change against a Business Service. Priority was set to Critical!');
}
```

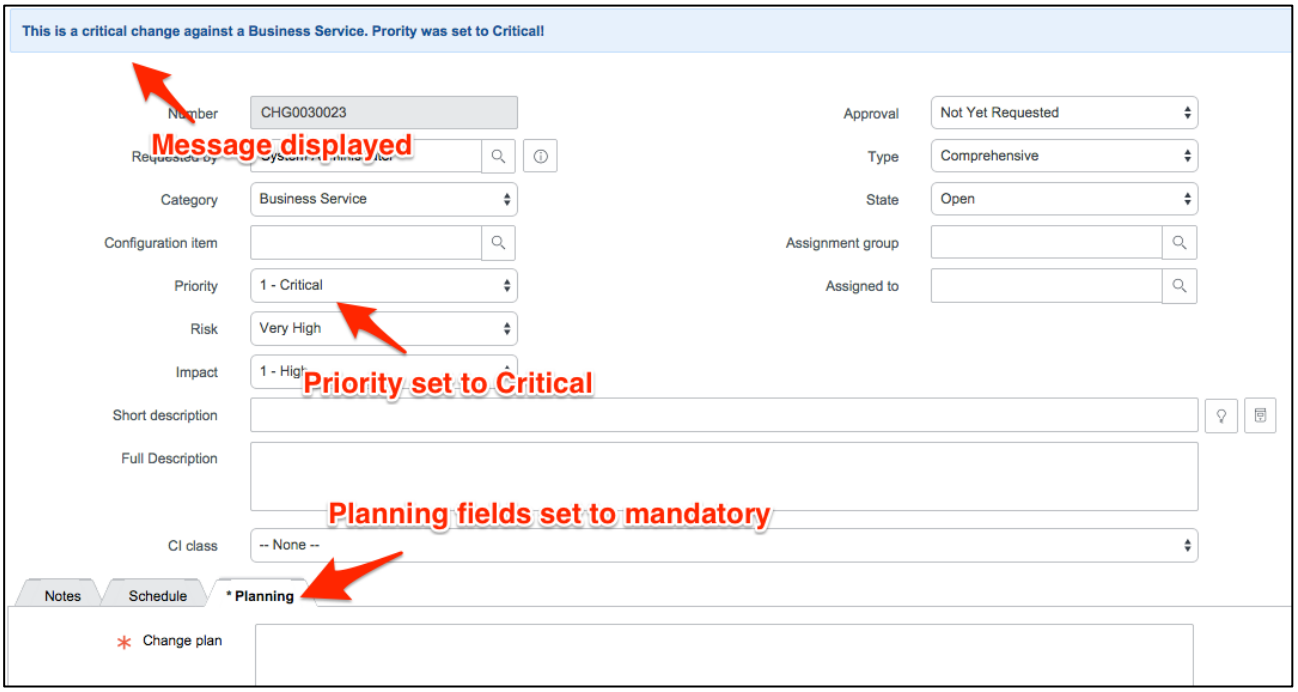
6. **Save** the UI Policy.

7. Add the following UI Policy Actions:

   Backout plan – **Mandatory = True**
   Change plan – **Mandatory = True**
   Test plan – **Mandatory = True**

Lab Success Verification

1. When creating a new Comprehensive Change that matches the conditions attached to the UI Policy, the Planning fields become mandatory and the Priority raises to **1 – Critical**.



Lab Goal

This lab explains how to fix a broken UI Policy and the related scripts using different Debugging techniques.

Lab 5.1 Debugging Client Scripting

Debugging Client Scripting

2. Open the "**K15 Debugging Client Scripts**" client script.
3. Select the **Active** check box to make the script active and save the **Client Script**.
4. In the script code, replace the string **<your initials>** in the **jslog** statement with your initials.
5. Run the **Syntax Checker**. Does it find any errors?
6. Update the script.
7. Launch the JavaScript Debugger Window by selecting the **cog** wheel on the upper-right side next to the logout button. Click **JavaScript Log and Field Watcher** at the bottom of the list.
8. Open an Incident with **Impact High** or **Medium** in order to trigger the Client Script.
9. Do you see a log message in the JavaScript Debugger window?
10. Open the **Developer Tool** of your browser.
11. Again open an Incident with **Impact High** or **Medium**. Make sure the page is reloaded so the script gets triggered again.
12. Do you see any error message in the console output of your browser?
13. Adjust the script based on the console output you saw.

14. Open an Incident with **Impact Low** in order to trigger the Client Script.

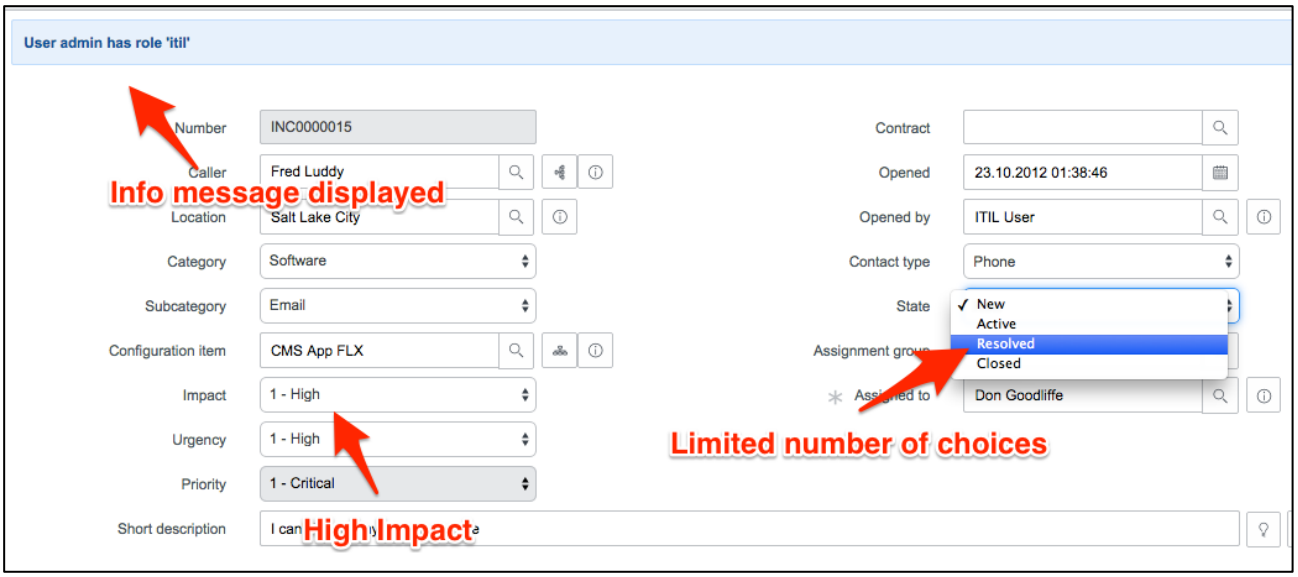
15. The **try catch block** will make sure errors get printed to the JavaScript log. Search for the output prefixed with **Debug>>>** and **your initials**.

16. Adjust the script and repeat steps 18 to 20 until there are no more errors printed to the JavaScript log.

Lab Success Verification

1. After successfully fixing the **Client Script**, there should be no errors logged to the JavaScript log and you should see the following.

   When opening a **High** or **Medium Impact Incident**:



   When opening a **Low Impact Incident**: