ServiceNow Scripting

Table of Contents

ServiceNow Scripting Overview

Module 01

Objectives

Define Scripting in ServiceNow

Determine when to use Scripting

Describe the different places ServiceNow scripts execute

Discuss the course contents

servicenow

In this module we will discuss the role scripting plays in ServiceNow.

When do I Script?

servicenow

- Extend functionality of ServiceNow beyond baseline
- You write a script to:
  - add functionality
  - extend Baseline functionality
  - integrate ServiceNow with 3rd party applications
  - automate processes

Examples

servicenow

- Update related records
  - Cascade a comment from a master incident to its children
  - Update Request ownership with a CI changes
- Approval Strategies
  - One or all doesn't meet your business requirement
  - Approvers need to be set dynamically
- Show/hide a form section
- Scan a list of CIs to dynamically determine risk based on criticality
- Recursively traverse the Business Service Map
- Query database
- Customize widgets
- Change default behaviors

The list of examples shown is representative of typical uses for scripting. There are, of course, no limits as to what one can do with JavaScript in ServiceNow.

When do I NOT script?

servicenow

- Make sure the script IS really needed:
  - –You have checked that the process you are implementing cannot be modified so scripting is not needed
  - –Can you get 90% of what you need without one?
  - –How business critical is the requirement?
  - –Will an ACL do what you need?
- ServiceNow is continually improving and what you scripted last year might not need to be scripted now
  - –Check the wiki
  - –Check release notes

APIs

servicenow

- ServiceNow provides APIs in the form of:
  - JavaScript classes
  - Web Services

JavaScript, popularized by Yahoo, is the most prevalent scripting language on the web. It is object-oriented, runs within a browser, and needs no license.

Script Execution in ServiceNow

servicenow

- Scripts execute:
  - Client forms (web browser)
    - Auto-populate a field based on a value in another field
    - Show/hide a section
  - Server
    - Modify a database record
    - Generate an event
  - MID server
    - Integrate to a 3rd party application

Where a script executes matters:
- Performance considerations
- Access to objects and methods
  - Client side has access to data on a form
  - Server side has access to database records

What is covered in this course?

servicenow

- Preparing to Script
- Client Scripts
- UI Policies
- Business Rules
- GlideSystem
- GlideRecord
- Managing Events
- Scheduled Jobs
- Workflow Scripting
- UI Actions
- Script Includes

Module Recap

servicenow

Core Concepts

Scripts extend the functionality of ServiceNow

The answer is not always to create a new script

ServiceNow uses JavaScript as its scripting language

Scripts execute in different locations

- Client
- Server

Real World Use Cases

Why you would use this

When you would use this

How often you would use this

Discuss: Why, when and how often would you use the capabilities shown in this module?

Preparing to Script in ServiceNow

Module 02

Objectives

Use the Script Editor features

Debug with the Syntax Checker

Personalize script lists

Customize the Edge

Know where to get scripting help

Labs

2.1 Using the Syntax Editor

2.2 Syntax Checking

2.3 Personalizing Lists

2.4 Using the Edge

2.5 Scripting Resources

servicenow

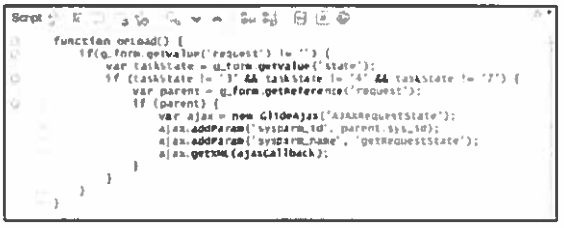
In this module you will prepare your ServiceNow instance for writing scripts.

Using the Script Editor

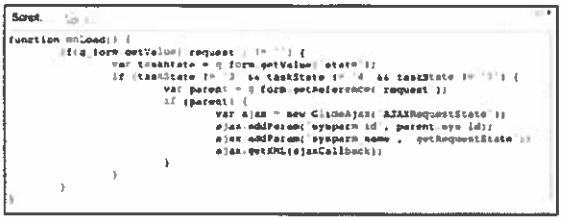
servicenow

The Script editor has the following features:

- Syntax Highlighting
- Layout Formatting
- Special character locating
- Script Macro (stub code) access
- Syntax Checking
- Code Block commenting/ uncommenting
- Search/Replace/Replace All
- Save
- Full screen editing mode



Script Editor Enabled



Script Editor Disabled

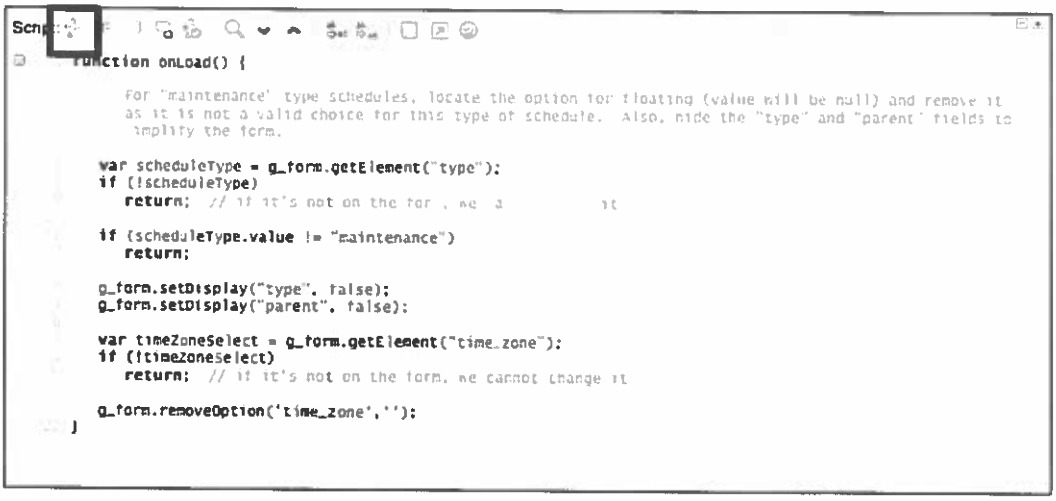
The Script editor is the default scripting editor for the Calgary release of ServiceNow. This editor requires the Syntax Editor plugin which is is enabled baseline for new instances. For upgraded instances, a system administrator must activate the plugin.

The Script editor can be disabled by users preferring to develop in an HTML text field. System administrators can disable the editor for all users regardless of user preference by setting the glide.ui.javascript_editor property to false.

Syntax Highlighting

servicenow

- Applies color coding to script for readability
  - Green = Comments
  - Purple = JavaScript Commands
  - Blue = Strings, Reserved Word

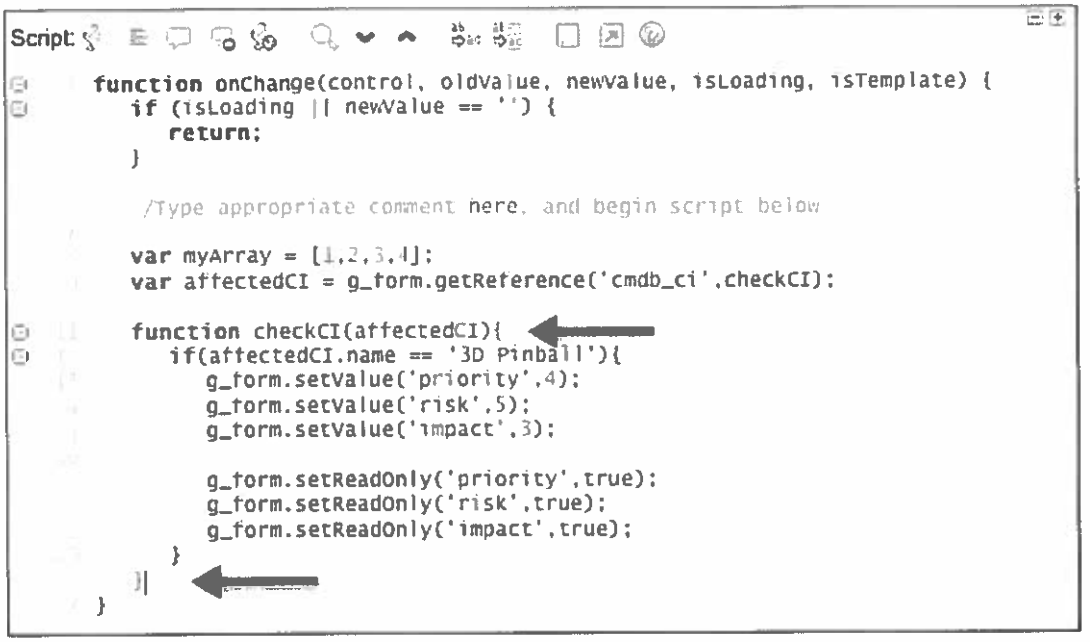


Select the Syntax Highlighting button to toggle the Syntax Editor on/off. When the Syntax Highlighting option is off, the script editor is a basic text field.

Special Character Highlighting

servicenow

- Highlights matching special characters: [ ], { }, ( )

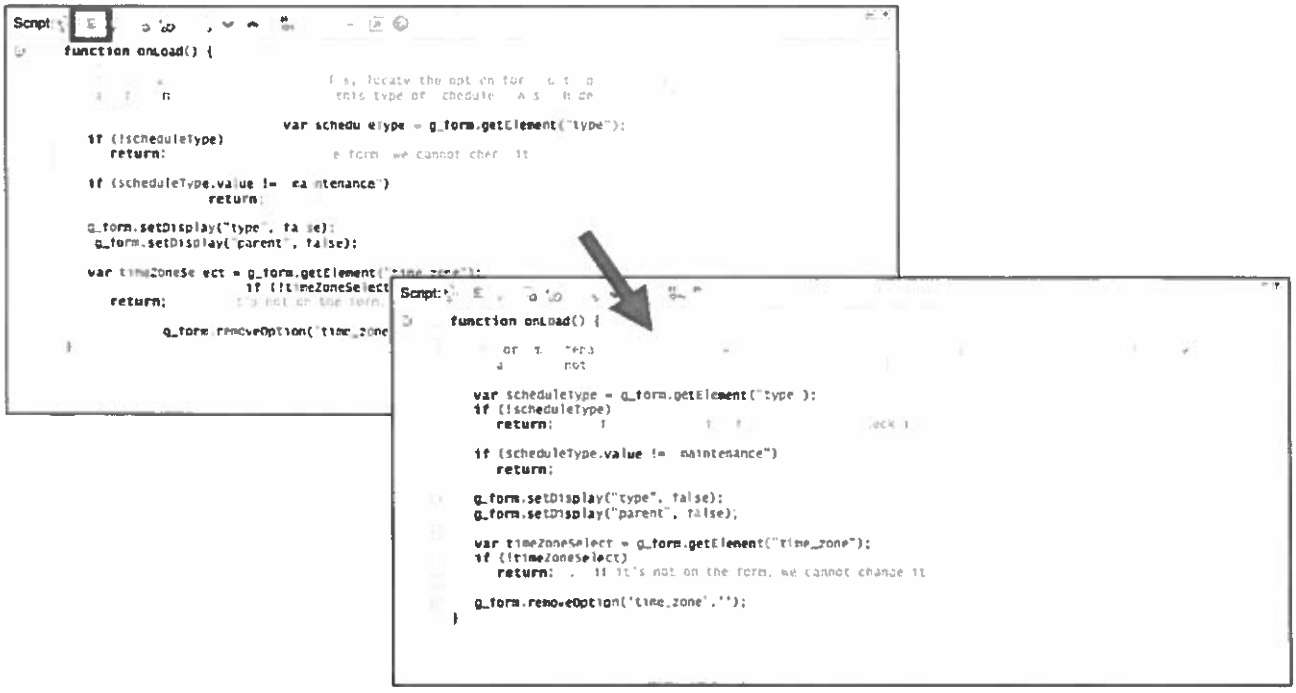


Place the cursor after any special character and the matching special character is automatically highlighted. This feature is useful for debugging.

Layout Formatting

servicenow

- Applies standard indentation for improved readability

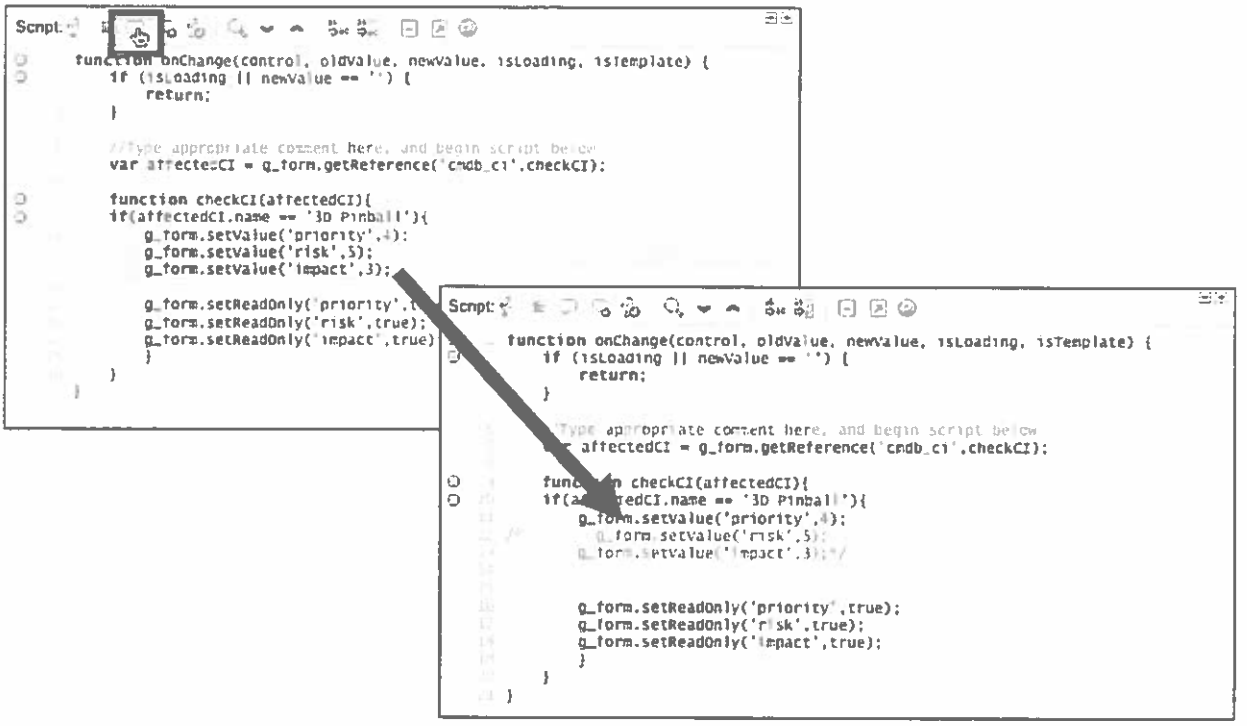


Select the Format Code button to automatically apply JavaScript standard indenting to your script.

Commenting/Uncommenting Code Blocks

servicenow

- Select and click to comment / uncomment



Use the **Comment Selected Code** option to comment out a block of code.

1. Highlight the code to be commented out.
2. Select the Comment Selected Code button.

Use the **Uncomment Selected Code** option to make code active within the script.

1. Highlight the commented code to be made active.
2. Select the Uncomment Selected Code button.

If a single line is selected, the single line comment characters (//) are prepended to the line. If multiple lines are selected, the comment is enclosed in /* and */ characters. Examples:

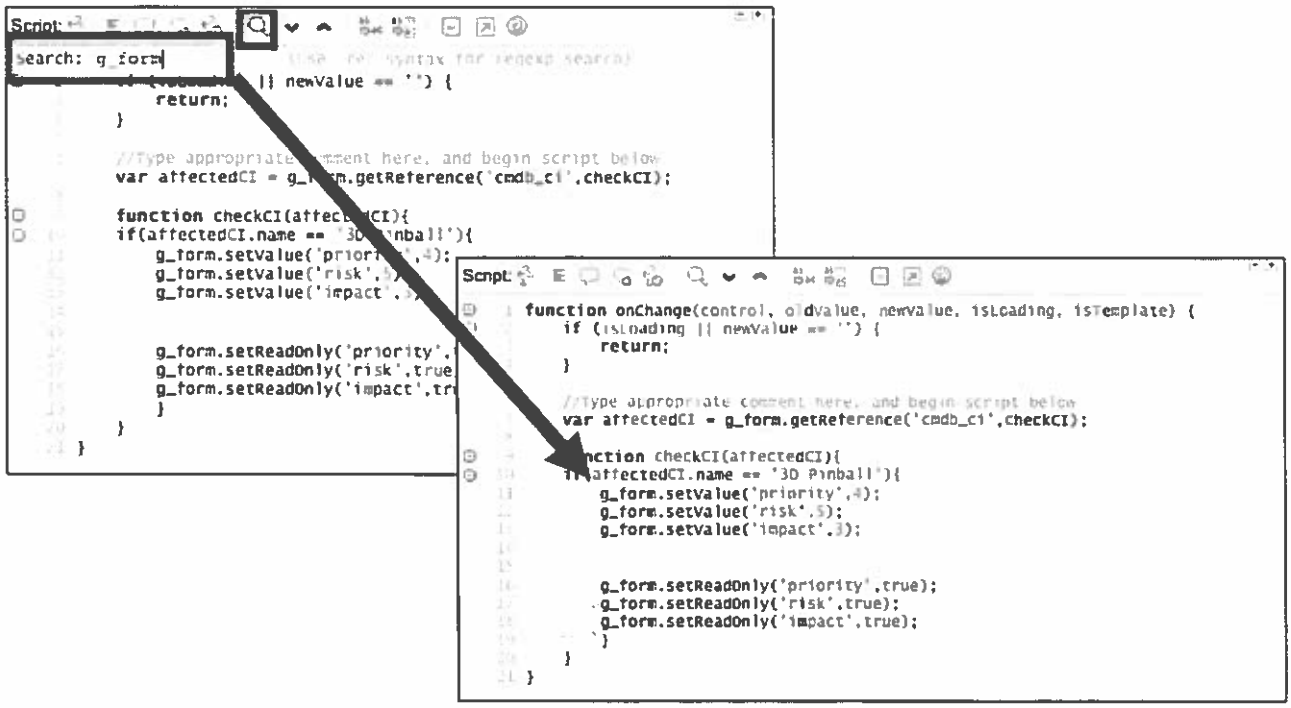
// This is a single line comment.

/*This is a
multi-line comment.*/

Searching

servicenow

- Search for strings or regular expressions



To find a string in the script editor,

1. Select the Search button.
2. Enter the text to search for in the Search field.
3. Press the <enter> key on the keyboard.

All instances of the search string are automatically highlighted. To move between occurrences of the search string, use the Find Next (down arrow) button and the Find Previous (up arrow) button.

The Search feature can also locate text specified with a regular expression. Regular expressions must be bracketed by / characters: / < your regular expression here> /

For example, the regular expression:

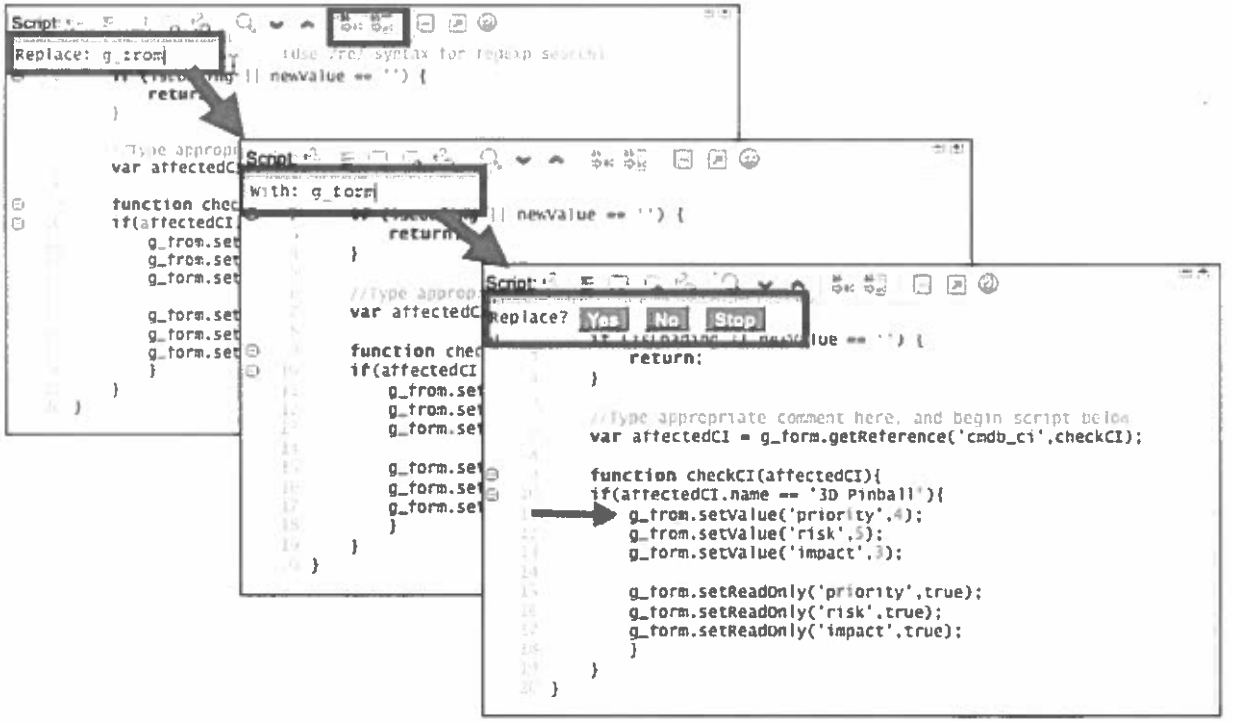
/g_[a-r]{4}/

would also find the string g_form.

Replace / Replace All

servicenow

- Replace a string with another string



To replace a string in the script editor using the **Replace** button:

1. Select the Replace button.
2. Enter the text to replace in the Replace field and press the <enter> key.
3. Enter the replacement text in the With field and press the <enter> key.
4. Select Yes to replace, or No to skip for each match. Select Stop to discontinue the replacement process.

To replace a string in the script editor using the **Replace All** button:

1. Select the Replace All button.
2. Enter the text to replace in the Replace field and press the <enter> key.
3. Enter the replacement text in the With field and press the <enter> key.

The Replace and Replace All features can also search for text specified with a regular expression. Regular expressions must be bracketed by / characters: / < your regular expression here> /

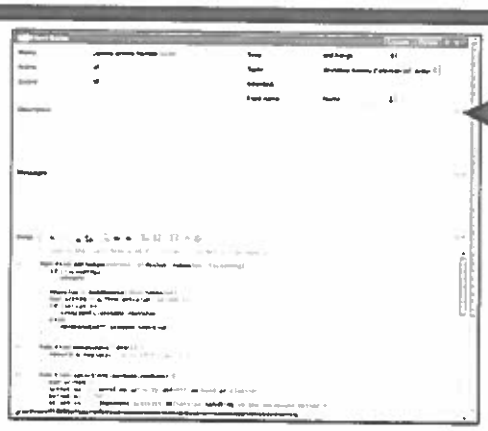
For example, the regular expression:

/g_[a-r]{4}/

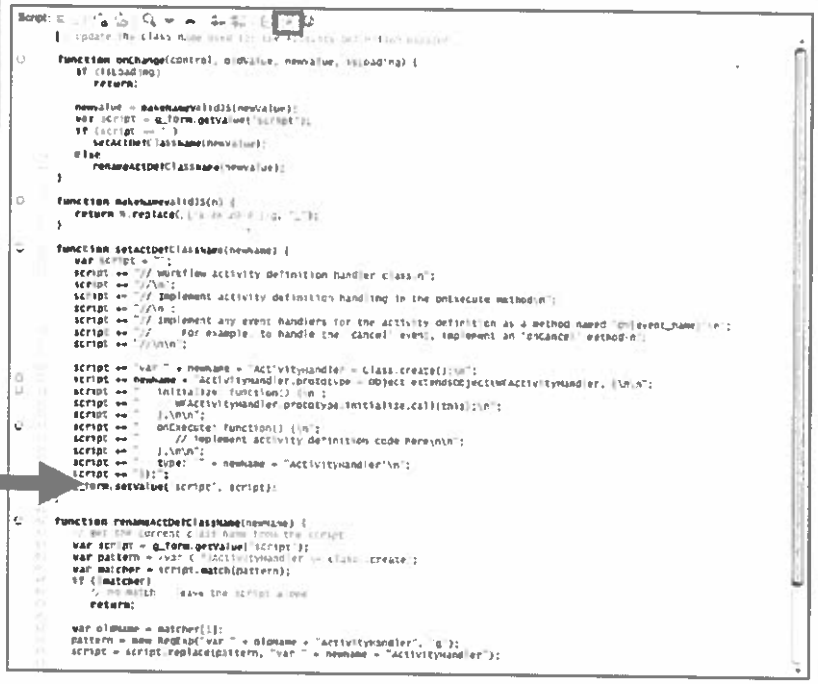
would find both the strings g_form and g_from.

Maximize Script Window

servicenow



This space is wasted when writing scripts



The entire frame is a JavaScript editor

Every script has two parts:
- Trigger (when to execute)
- Script (what to do)

Once configured, the trigger is seldom changed. Scripts under development change frequently. To have more space for the script, use the Toggle Full Screen button to maximize the script editor. Select the Toggle Full Screen button again to restore the script editor to its default size.

Script Macros

servicenow

- Shortcuts to commonly used syntax
- Insert stub code into a script
- Type the macro name followed by <tab>
- Default macros include:
  - for: inserts a standard for loop
  - doc: inserts a multi-line comment
  - help: inserts a list of macros and their shortcuts

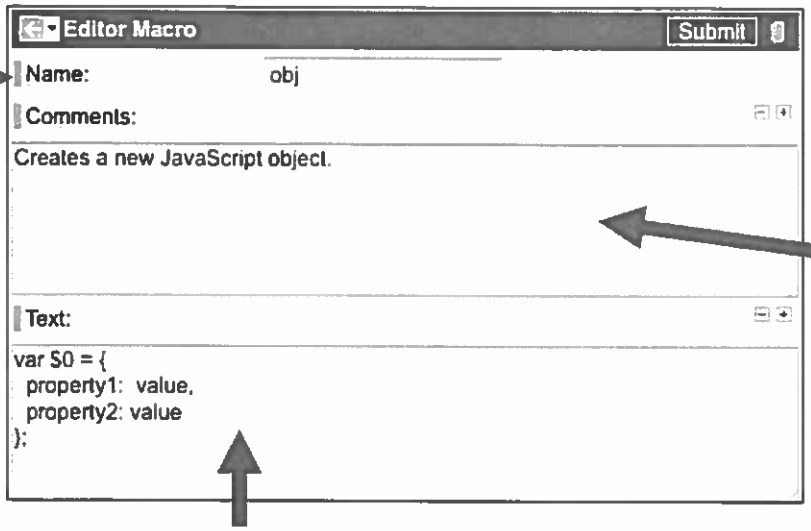
For example, in the JavaScript editor, typing the word **for** followed by the **<tab>** key results in the following insertion:

```
for (var i=0; i< myArray.length; i++) {
  //myArray[i];
}
```

Creating Script Macros

servicenow

Macro Name



Comment that is displayed when the help macro is used

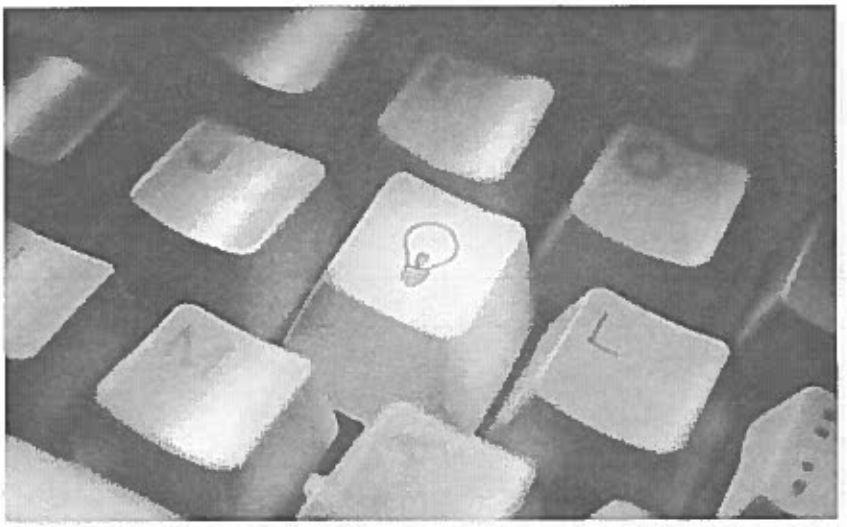
Macro text: $0 indicates the cursor position when the macro is used

Preparing to Script

To launch the Syntax Editor Macros go to **System Definition > Syntax Editor Macros.**

Lab

servicenow



2.1 Using the Syntax Editor

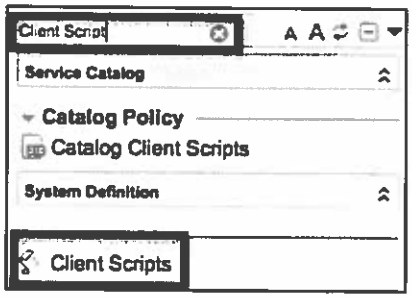
Lab Goal

In this lab you will practice with the Script Editor in preparation for writing scripts in ServiceNow.

Lab 2.1 Using the Script Editor

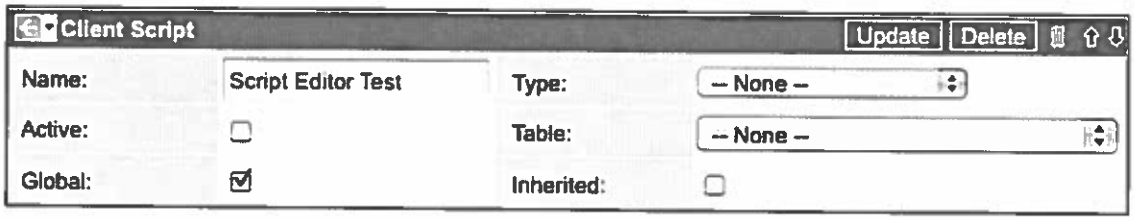
Using the Script Editor

1. In the **Application Navigator**, type **Client Script** in the **Type filter text** box.



2. Open the **System Definition > Client Scripts** module.

3. Create a new Client Script called Script Editor Test. De-select the Active check box to prevent the script from running.



4. In the Script Editor enter the following text:

```
Code to test the Script Editor Features:
 - Syntax Highlighting
 - Special Character Highlighting
 - Layout
 - And More!
```

5. Convert the text to a comment.

   a. Select all of the text .

   b. Select the Comment Selected Code button ().

6. In the Script Editor, below the comment, enter the following script. As you enter the script, note the syntax highlighting, special character highlighting, and indentation.

```
var myNum = 42;
var mySring = "Hello World";
var myArray = ["iPhone","Android","Blackberry","Window Phone"];
var myObject = {
   property1:  "first",
   property2:  "second",
   property3:  "Third"
};

for(var i = 0 ; i < myArray.length ; i++) {
   alert("The current value of myArray is: " +myArray[i]);
}
```

7. Save the script by selecting the Save button ().

8. Turn off syntax highlighting by selecting the **disable syntax highlighting** icon (). What differences do you notice in the script editor?

9. Turn syntax highlighting on.

10. Use the Replace All feature to change the value for myObject.property3 value from Third to third.

    a. Select the Replace All button ().

    b. Type Third in the Replace field and press the <enter> key.

    c. Type third in the With field and press the <enter> key.

11. Use the Replace feature to change the 2 in myObject.property2 to myObject.propertyTwo.

    a. Select the Replace button ().

    b. Type 2 in the Replace field and press the <enter> key.

    c. Type Two in the With field and press the <enter> key.

    d. Select the No button when prompted to replace the 2 in 42.

    e. Select the Yes button when prompted to replace the 2 in property2.

12. Open the **System Definition > Syntax Editor Macros** module.

13. Create a new macro with these parameters:

```
Name: try
Comment: try/catch
Text: try{
        $0
      }
      catch(err){
       jslog("A runtime error occurred: " + err);
      }
```

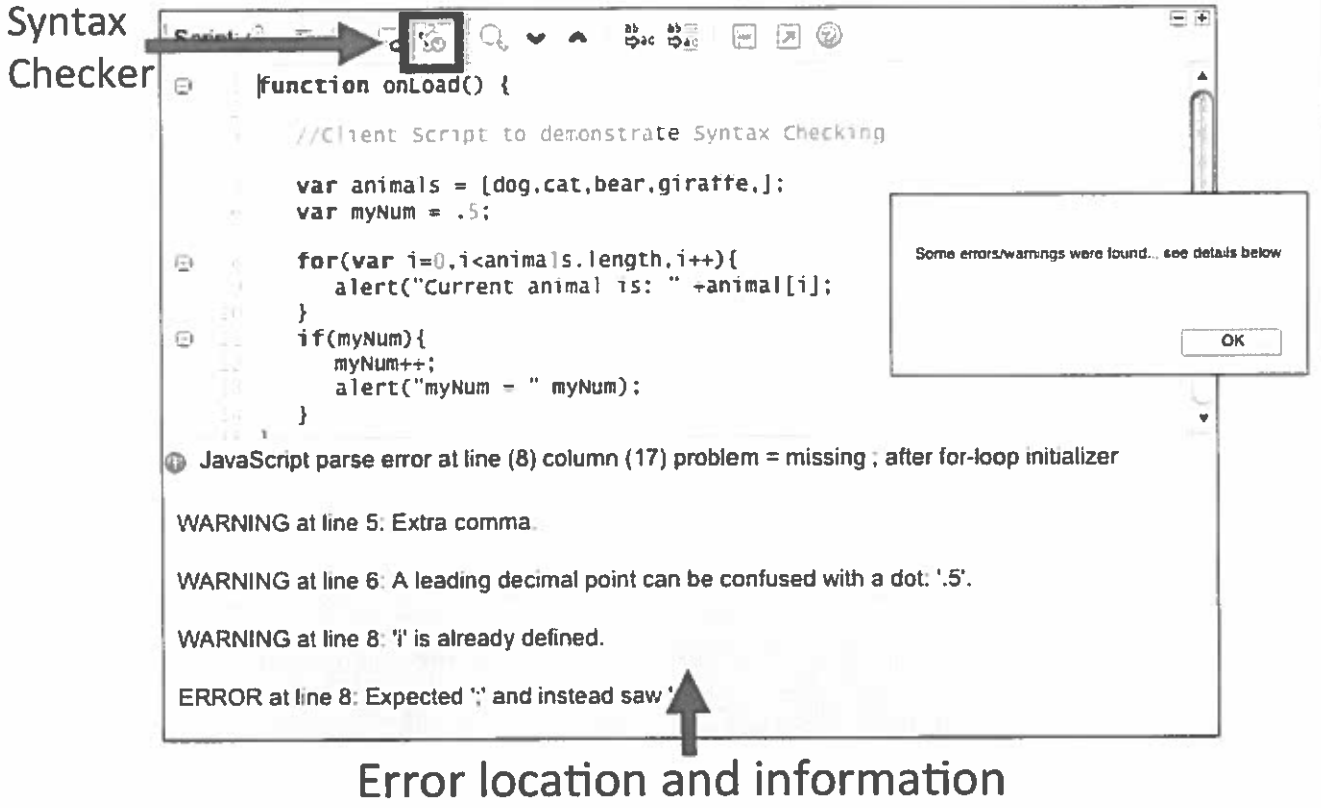
14. Save.

15. Open the Script Editor Test Client Script.

16. Test the try/catch macro by typing **try** into the editor and pressing **<tab>**. Did the macro work? Is the cursor where you expected it to be?

Using the Syntax Checker

servicenow



If you cannot locate an error on the line called out by the Syntax Checker, look at the preceding line(s) of code.

If there are severe errors such as the JavaScript parse error shown above, the script will not save until the error is corrected.

Syntax Checker

servicenow

- The Syntax Checker finds basic JavaScript errors such as:
  - Missing characters like [and }
  - Missing ; at the end of JavaScript statements
  - Incomplete arguments in for loops
  - Bad function calls
- Does not find:
  - Typos in variable names
  - Typos in function calls
  - Typos in method calls
- Cannot determine if your script works as expected

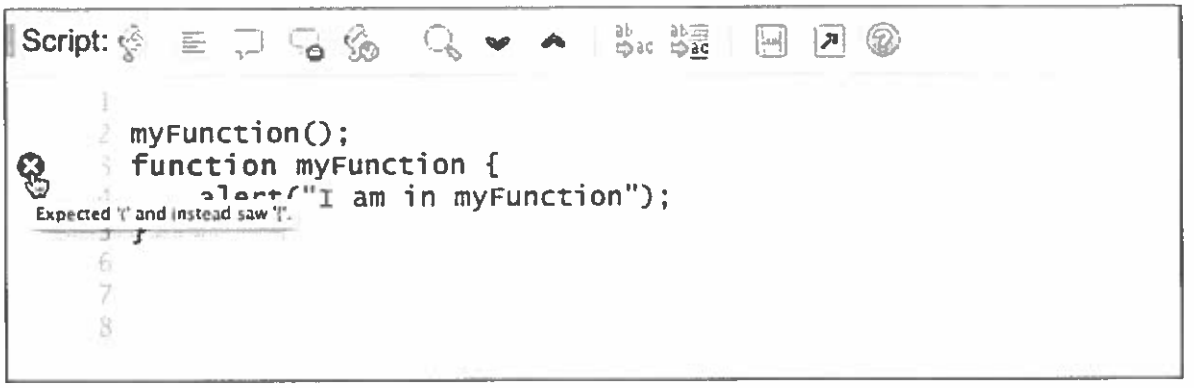
As with all programming editors and tools, the Syntax Checker cannot find all errors in a script. *Any valid JavaScript will pass the Syntax Check* even if the code does not do what you intended.

Although the Syntax Checker cannot find all errors, you should always run it. Some script types will save with syntax errors (Workflow scripts) and others will not.

Real Time Syntax Checking

servicenow

- Real time error checking as you script
  - Red circles with white Xs indicate errors
  - Yellow triangles with black !s indicate warnings
- System administrator must enable/disable
  - Turned on/off for all users simultaneously
  - Not user settable



System administrators must edit the glide.ui.syntax_editor.show_warnings_errors property to enable real time syntax checking. There are three possible values:

- HIDE_ALL (default): disables real time syntax checking
- ERRORS_ONLY: enables real time syntax checking for errors
- SHOW_ALL: enables real time syntax checking for errors and warnings

Lab

servicenow

2.2 Syntax Checking

Lab Goal

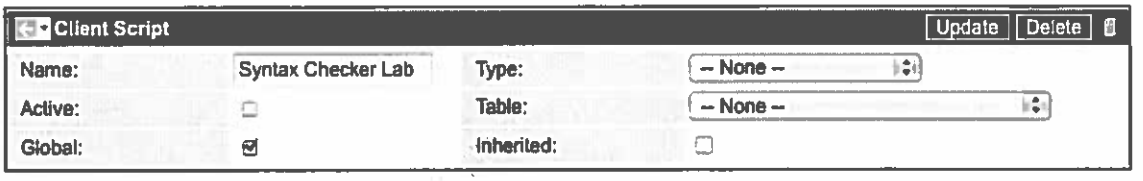
In this lab you will practice with the Syntax Checker to determine which types of errors the checker traps.

Lab 2.2
Syntax Checking

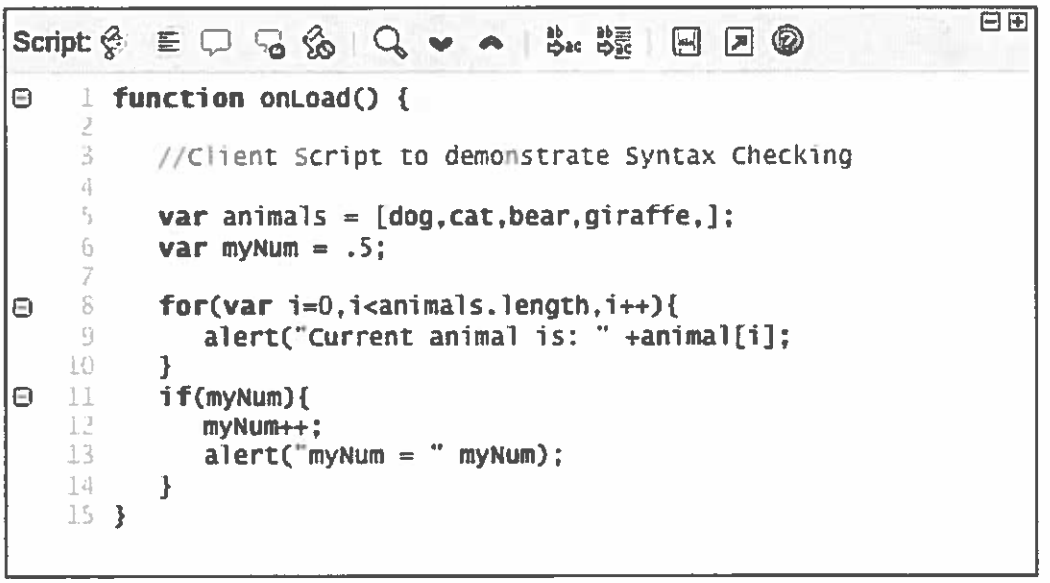
Syntax Checking in the Script Editor

1. Create a new Client Script called **Syntax Checker Lab.**

2. De-select the Active checkbox to prevent the script from running.



3. Enter the following script (including errors) into the Script editor:



4. Study the script and without running the Syntax Checker, identify the errors in the script. Mark the errors on the screenshot in step 3. (Hint: There are 6 types of errors.)

5. Try to save your script.

    a. What does the flashing yellow bar mean?

    b. Was your script saved? How can you tell?

6. Run the Syntax Checker. Correct only the errors found by the Syntax Checker. Run the Syntax Checker again.

7. Repeat step 6 until no syntax errors remain.

8. Did the Syntax Checker find all the errors in the script? Which error(s) were not found?

9. Save the Client Script.

Enabling and Using Real Time Syntax Checking

1. In the Application Navigator, type sys_properties.list in the Type filter text field. (Do not press the Enter/Return key after typing sys_properties.list.)

2. Search for the property glide.ui.syntax_editor.show_warnings_errors.

3. Change the Value for the glide.ui.syntax_editor.show_warnings_error property to SHOW_ALL.

4. Select the **Update** button.

5. Open the Syntax Checker Lab Client Script.

6. Remove the final quote mark from the alert on line 13.

7. Does the symbol to the left of the line number indicate that you have a warning or an error?

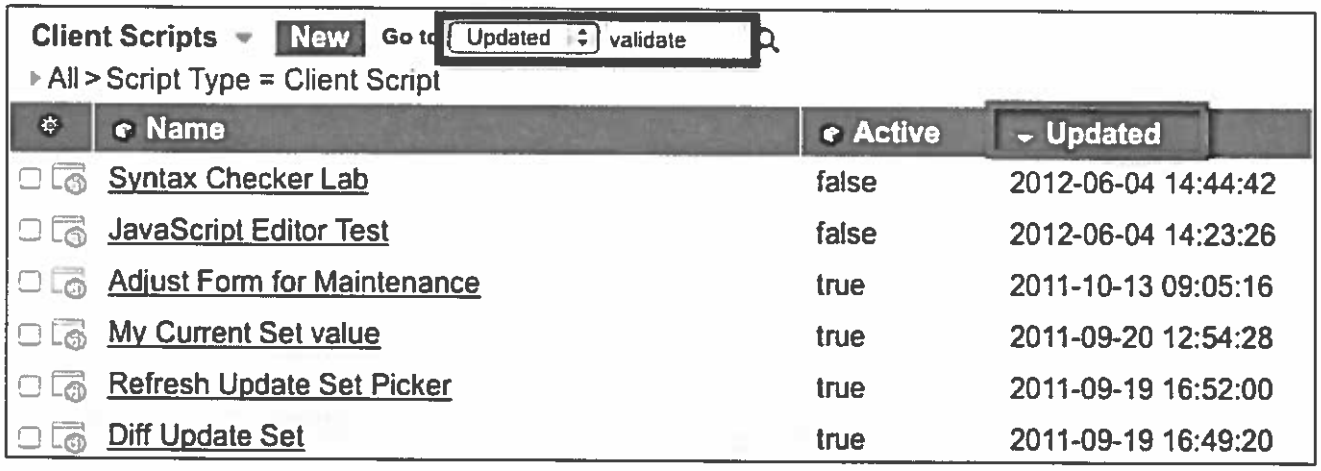
8. Hover the mouse over the symbol. Does the message accurately reflect the problem?

9. Fix the problem on line 13 and insert other errors into your Syntax Checker script one at a time. Examine the resulting real time errors and warnings as they appear.

Locating Scripts

servicenow

- Search by name in the list
- Sort the list by Updated date in order to find the most recently edited scripts (must be configured)
- Sort the list by Updated by (must be configured)



Name	Active	Updated
Syntax Checker Lab	false	2012-06-04 14:44:42
JavaScript Editor Test	false	2012-06-04 14:23:26
Adjust Form for Maintenance	true	2011-10-13 09:05:16
My Current Set value	true	2011-09-20 12:54:28
Refresh Update Set Picker	true	2011-09-19 16:52:00
Diff Update Set	true	2011-09-19 16:49:20

Each script type, such as Client Script or Business Rule, has its own list view. Use the list to locate scripts. You can search by name or by other criteria displayed in the list view. For example, to search for a script with the name validate, type validate in the search field and select the search icon.

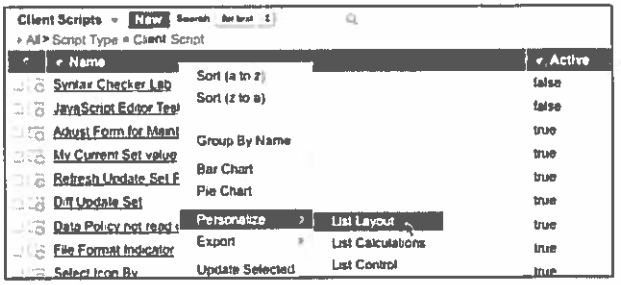
Some helpful search strategies:

- *mySearchString does a "contains" search
- mySearchString (no leading or trailing characters) does a >= search (like starts with but everything after the search string alphabetically also)
- mySearchString% does a "starts with" search
- %mySearchString does an "ends with" search

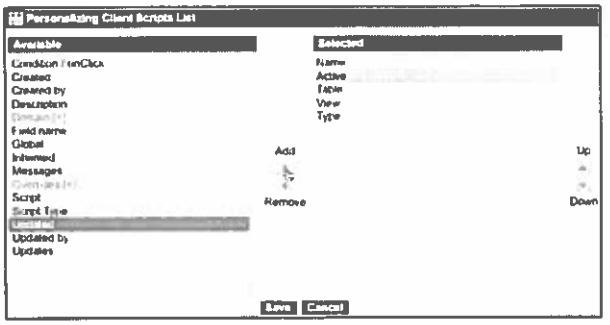
It is useful to add the Updated column to the list view to quickly locate the most recently edited scripts. This strategy is especially useful in class because you will be the only user editing scripts on your instance. Your scripts will be the most recently edited. The Updated column is not displayed by default and must be added to the list.

Adding the Updated Column

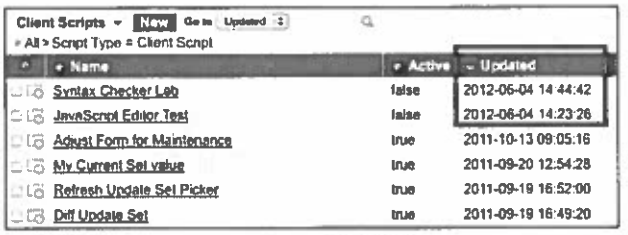
servicenow



1. Personalize the List Layout



2. Add the Updated column from the slushbucket



3. Sort by the Updated date

Personalizing a list's layout changes the layout for all users. An alternate strategy that modifies only the current user's list is to personalize using the Personalize List icon (gear icon) in the leftmost column of the list header.

Lab

servicenow

2.3 Personalizing Lists

Lab Goal

In this lab you will add the Updated column to the list in order to quickly locate the most recently edited scripts. You will choose whether to add the personalization for all users or for the currently logged in user only.

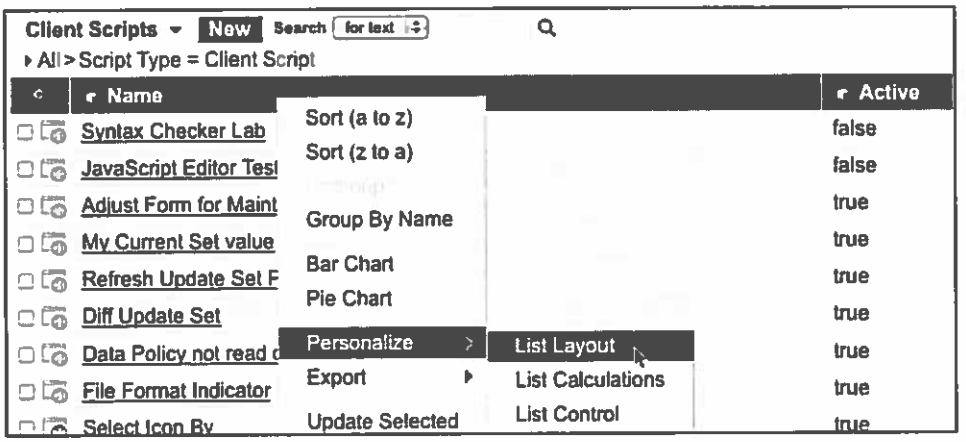
Lab 2.3 Personalize Lists

Personalize List - All Users

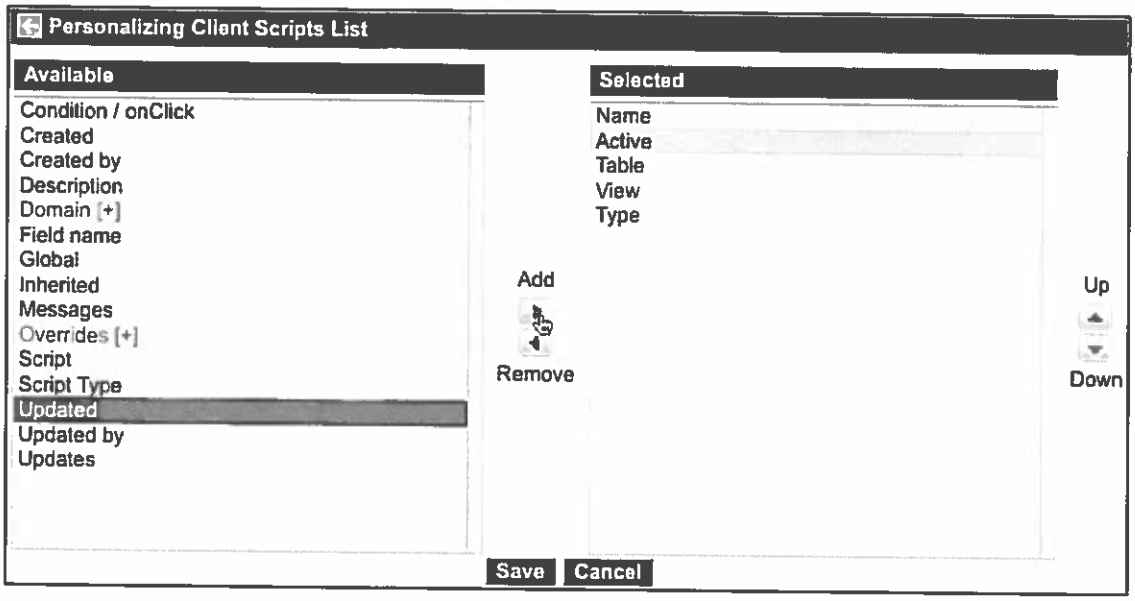
1. The steps in this section personalize the list for all users of the instance. To personalize the list for only the currently logged in user, skip this section of the lab and go to the Personalize List – Current User Only section.

2. Open **System Definition > Client Scripts.**

3. Right-click on the header bar and select **Personalize > List Layout.**



4. Select the **Updated** column from the slushbucket and use the **Add** button to add it to columns list.



5. If desired, use the **Up/Down** buttons to change the column's location.

6. Select the **Save** button.

7. Skip to the Testing section of the lab.

Personalize List - Current User Only

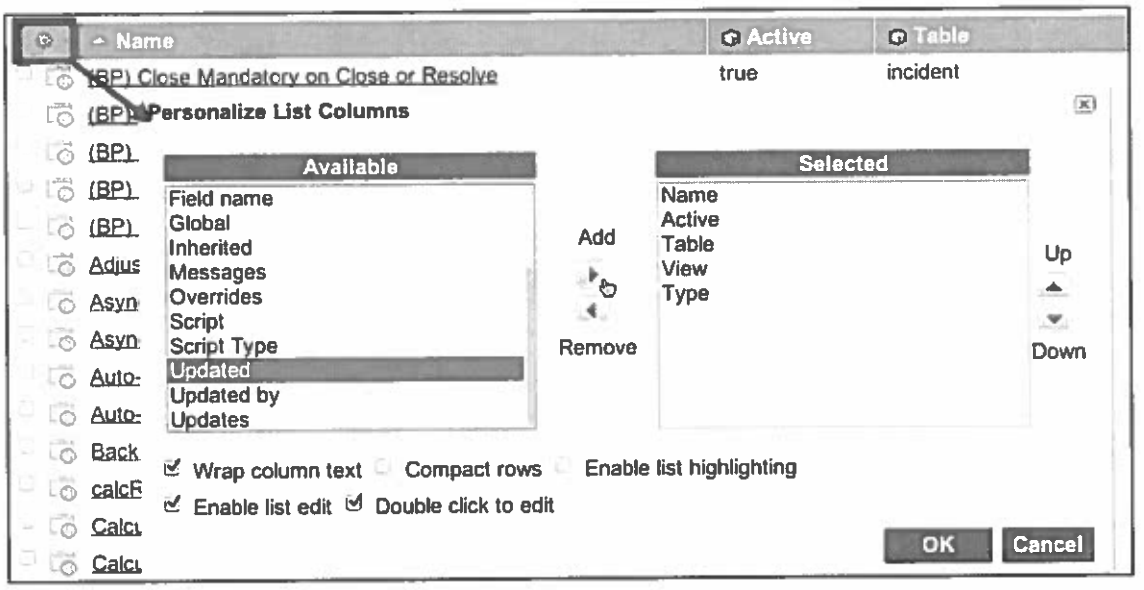
1. If you personalized the Client Scripts list for all users, skip this section of the lab and go to the Testing section.

2. Open **System Definition > Client Scripts.**

3. Select the **Personalize List Layout** icon (gear icon) from the leftmost column of the list header bar.

4. Select the **Updated** column in the Available slushbucket.

5. Select the **Add** button to add the Updated

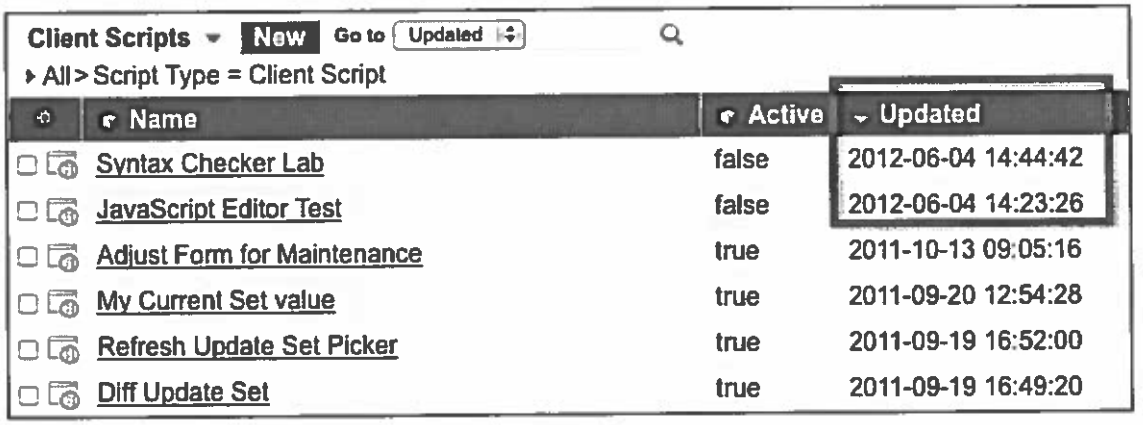


6. Select the **OK** button to save the change.

7. Why did the Personalize List Layout icon change on the list? Explain your reasoning here:

Testing

1. In the list view, click the Updated column header to sort the view based on update date.



2. Is the list sorted in ascending or descending order?

3. Click the Updated column header again. Did the sort order change?

What is the Edge?

servicenow

- The Edge is an optional toolbar used to:
  - Show or hide the application navigator
  - Show or hide the banner frame
  - Split the screen into panes
  - Create and manage bookmarks (favorites)
- Configured on a per user basis
- Efficiency tool for quick navigation

The following keyboard shortcuts are available for the Edge.

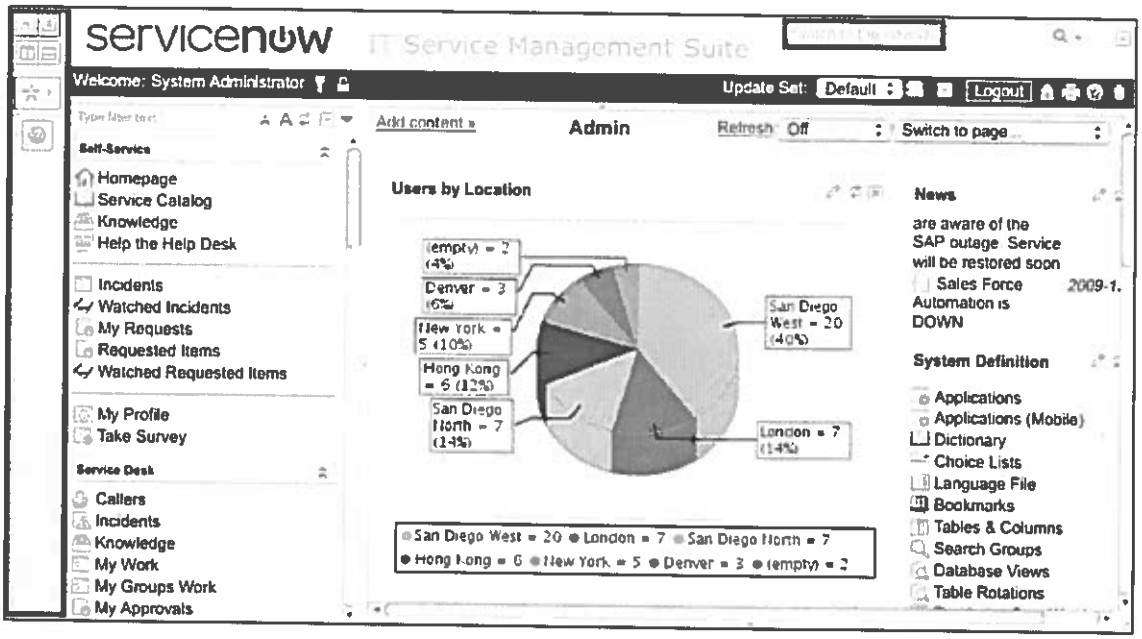
Access Key + N: Toggle navigator
Access Key + B: Toggle banner
Access Key + V: Toggle vertical split
Access Key + H: Toggle horizontal split
Access Key + M: Maximize the current pane
Hold CTRL (Windows) or COMMAND (Mac) and click to open the bookmark in a flyout.
Hold SHIFT and click to open the bookmark in a pane.

NOTE: Access keys depend on the browser and operating system you are using. For example, Google Chrome for Mac users press CTRL+OPT+N to toggle the navigator and Firefox for Windows users press ALT+SHIFT+N. The correct access keys for your browser are listed in the default help bookmark.

Enabling the Edge

servicenow

- Enable the Edge by selecting the Switch to the new UI link

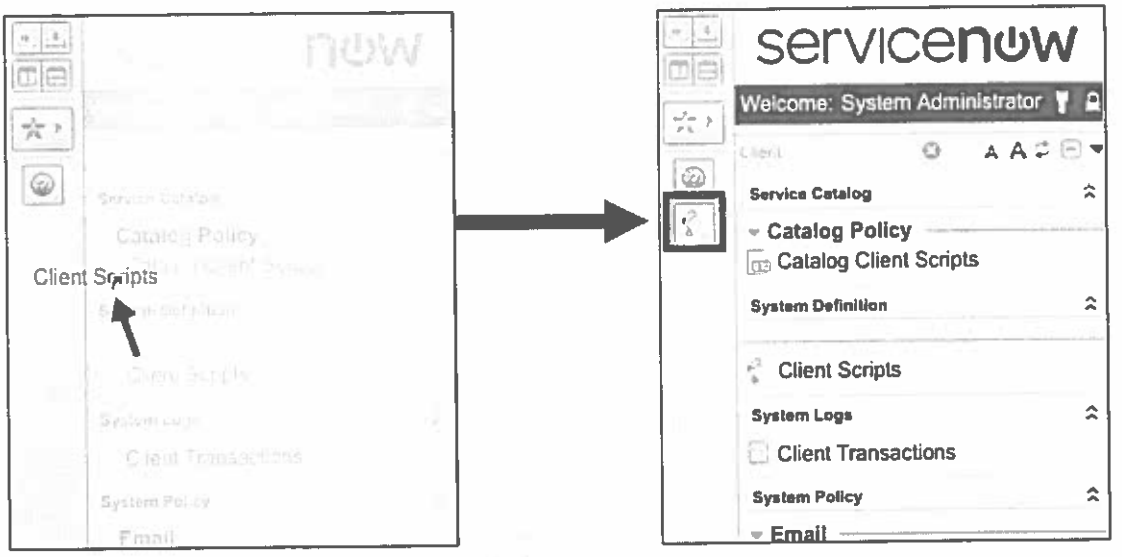


The Edge is disabled by default and must be enabled on a per user basis. If the link to enable the new UI is not available this feature has been disabled by a system administrator or your instance does not use the UI11 plug-in. If your administrator cannot or will not enable UI11, there is a workaround using labels. See the wiki for more information.

Populating the Edge

servicenow

- Click, hold, and drag applications to the Edge to add bookmarks



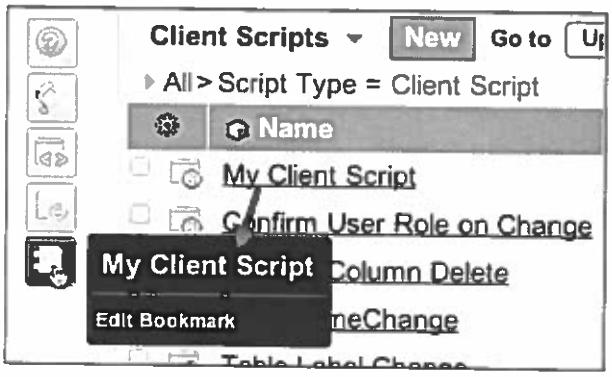
To remove an application from the Edge:

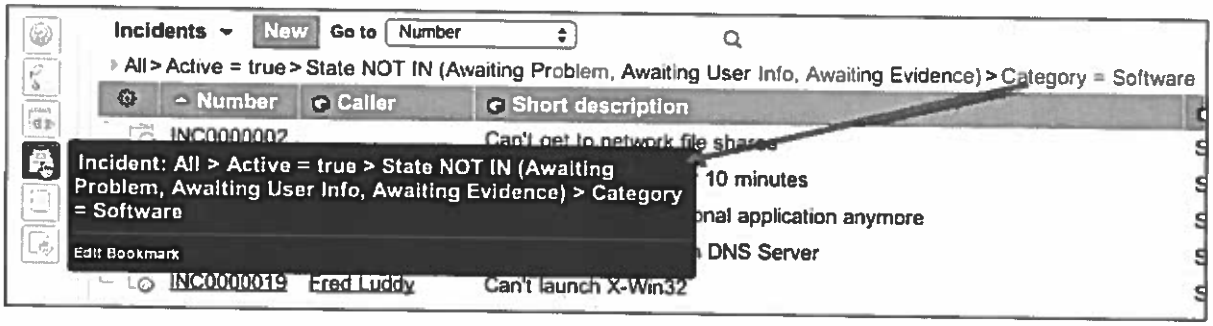
1. Hover your cursor over the bookmark and select Edit Bookmark.
2. Click the Delete button.

Context and the Edge

servicenow

- Context is preserved
  - Records
  - Lists
  - Filters





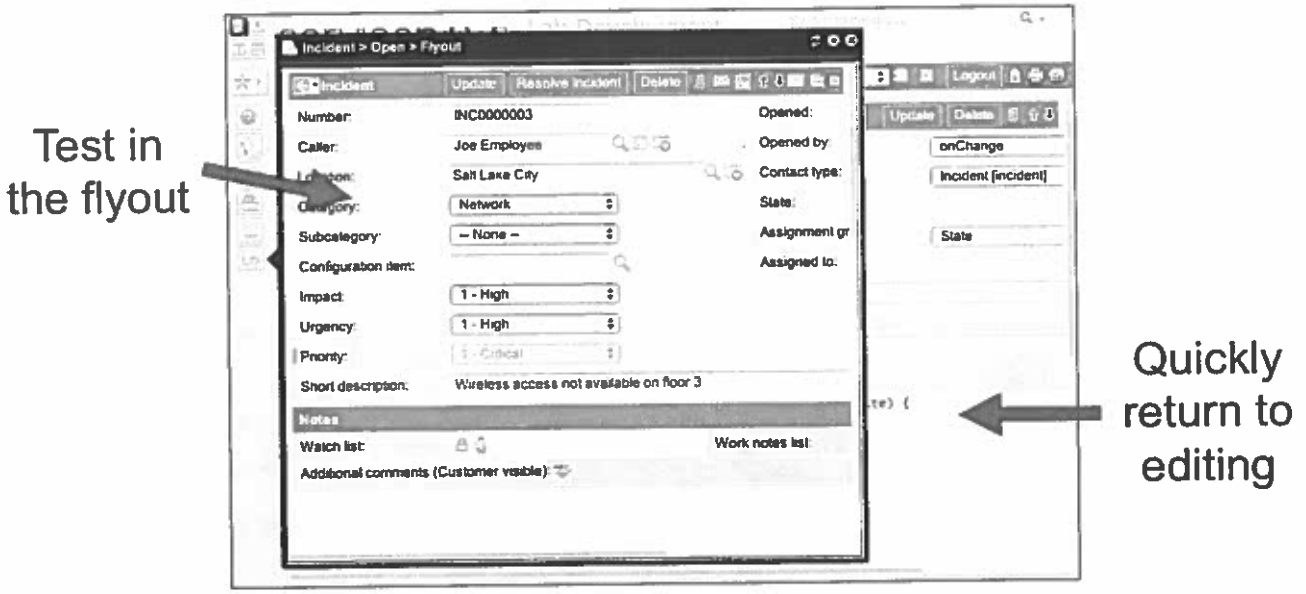
Context means that exactly what you drag to the Edge is what is opened when you select the Edge button. Dragging a specific Client Script to the Edge creates a button that opens that Client Script for editing. Dragging a module, such as Client Scripts, to the Edge opens the list of Client Scripts.

Adding elements you work with frequently to the Edge improves efficiency when writing and testing scripts.

Using Flyouts

servicenow

- Flyouts open forms and lists in an informational window without closing the pane you are working in
- Useful for testing when developing scripts



Follow these steps to make an Edge button a flyout:

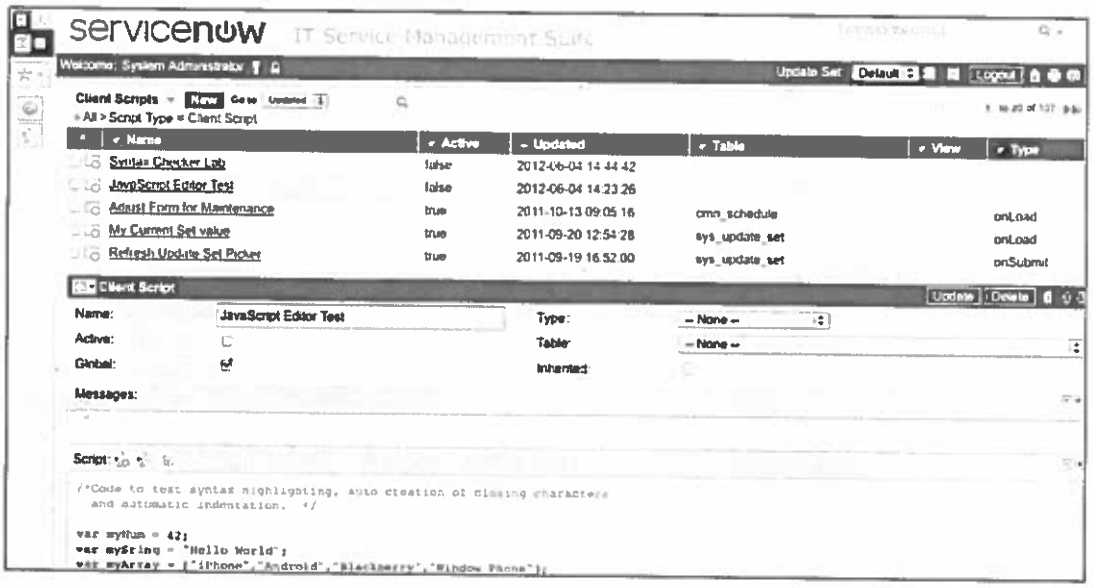
1. Hover over the button on the Edge.
2. Select the Edit Bookmark link.
3. Select the Flyout option.
4. Select the Update button.

When testing client side scripts, reload the client side script logic using the Refresh button in the Flyout header.

Using the Split Pane

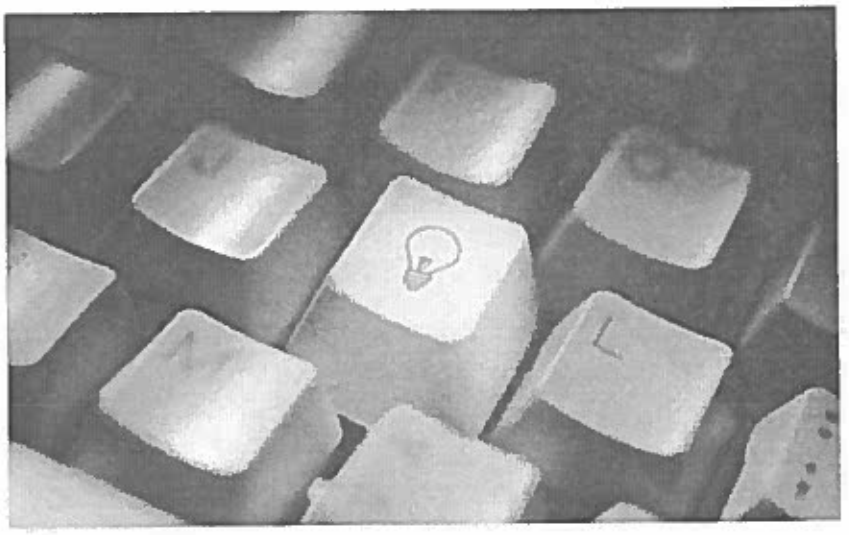
servicenow

- The Split Pane is a useful tool for working with lists
  - List in the top or left hand frame
  - Selected item's form in the lower or right hand frame



Lab

servicenow



2.4 Using the Edge

Lab Goal

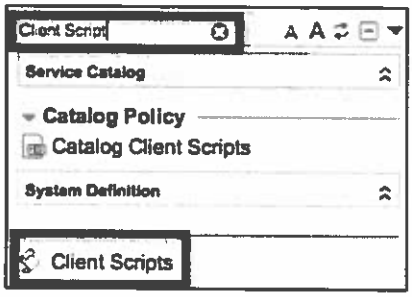
In this lab you will **enable** and begin to **populate** the ServiceNow **Edge**.

Lab 2.4
Using the Edge

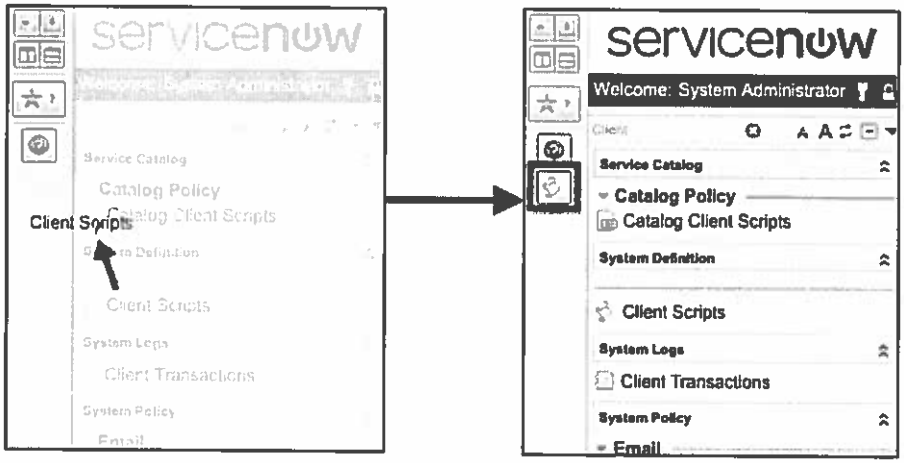
Using the Edge

1. In the ServiceNow banner, select the **Switch to the new UI** link. The Edge opens on the left side of the browser. Close the help pop out.

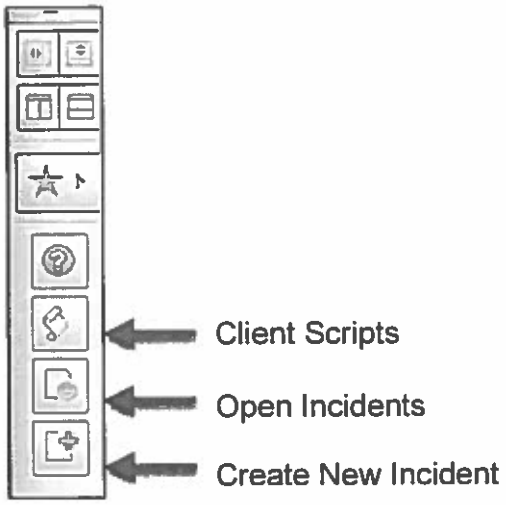
2. In the Application Navigator, type **Client Script** in the Type filter text box.



3. Click, hold, and drag the Client Scripts module. Drop it on the Edge.



4. Add the Open Incidents and Create New Incident applications to the Edge.



5. Close the Application Navigator by selecting the hide application pane button or by clicking the Close Pane arrow.

6. Make the Open Incidents button a Flyout.

   a. Hover your mouse over the Open Incidents button.

   b. Select the **Edit Bookmark** link.

   c. Select the **Flyout** option.

   d. Append the word Flyout to the button's Title.

   e. Select the **Update** button.

7. Use the bookmarks on the Edge to switch between Client Scripts, a list of open incidents, and creating a new incident.

8. Enable the Split Pane view by selecting the **Toggle horizontal split fom pane** button.

    a. Use the Edge to open the incident list.

    b. Use the list pane to select several incidents for viewing in the form pane.

9. Configure your ServiceNow instance using your preferred settings from this lab.

JavaScripting Information

servicenow

- Web Sites
  - w3schools.com
  - developer.mozilla.org/en/JavaScript
  - lynda.com (requires a paid subscription)
  - www.codeacademy.com
- Books
  - McFarland, David Sawyer. *JavaScript and jQuery: The Missing Manual*. Sebastapol, CA: O'Reilly, 2011
  - Crockford, Douglas. *JavaScript: The Good Parts. Sebastapol, CA: O'Reilly, 2008*

Scripting in ServiceNow requires a solid foundation in JavaScript including:
- JavaScript Data Types
- Methods
- Functions
- Conditional Logic
- Looping
- Debugging

There are many resources available for working with JavaScript.

ServiceNow Scripting Resources

servicenow

- community.servicenow.com
- wiki.servicenow.com
- User Groups
- Blogs
  - community.servicenow.com/people/SlightlyLoony/blog
  - community.servicenow.com/people/andrew.kincaid/blog
  - www.servicenowguru.com
  - www.john-james-andersen.com/

Membership in the ServiceNow community is free! Click the Create New Account link at community.servicenow.com.

Don't want to create an account? You don't have to but you will miss out on these features:

- Post questions on our Forums
- Comment on our Blogs
- Rate content
- Register for Local Events
- Join Local User Groups
- Get to know ServiceNow customers, partners, and employees
- Be an IT Hero

ServiceNow User Groups (SNUG)

servicenow

- Join a SNUG near you to meet other ServiceNow users in your local area: community.servicenow.com/community/user-groups

SNUGs are managed by ServiceNow users just like you. If there isn't a SNUG in your area, you can create one!

Lab

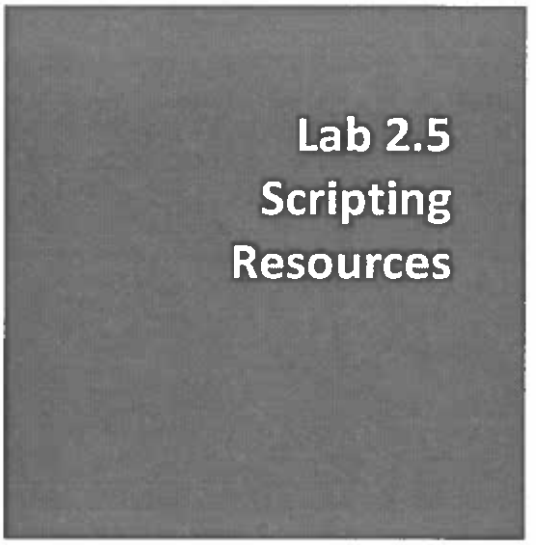
servicenow

2.5 Scripting Resources

Lab Goal

In this lab you will practice using the ServiceNow community to:

- Create an account
- Join a ServiceNow User Group
- Find Scripting Information
- Download a Book



Identifying Resources

1. In a web browser, navigate to community.servicenow.com.
2. Log in. If you don't have an account, create one by selecting the **Register** link. After requesting an account, continue with the lab.
3. Join the ServiceNow User Group (SNUG) nearest you.
    a. Select the **User Groups** link.
    b. Select the link to your geographical region.
    c. Select a SNUG from the list.
    d. Select the **Join this Group** button.
4. See what's new in Support.
    a. Select the **Support** link.
    b. Scroll down to the Recent Support Discussions section.
    c. Open **Filter by Categories & Tags**.
    d. Select a category that is of interest to you.
    e. Read a support posting from your category.

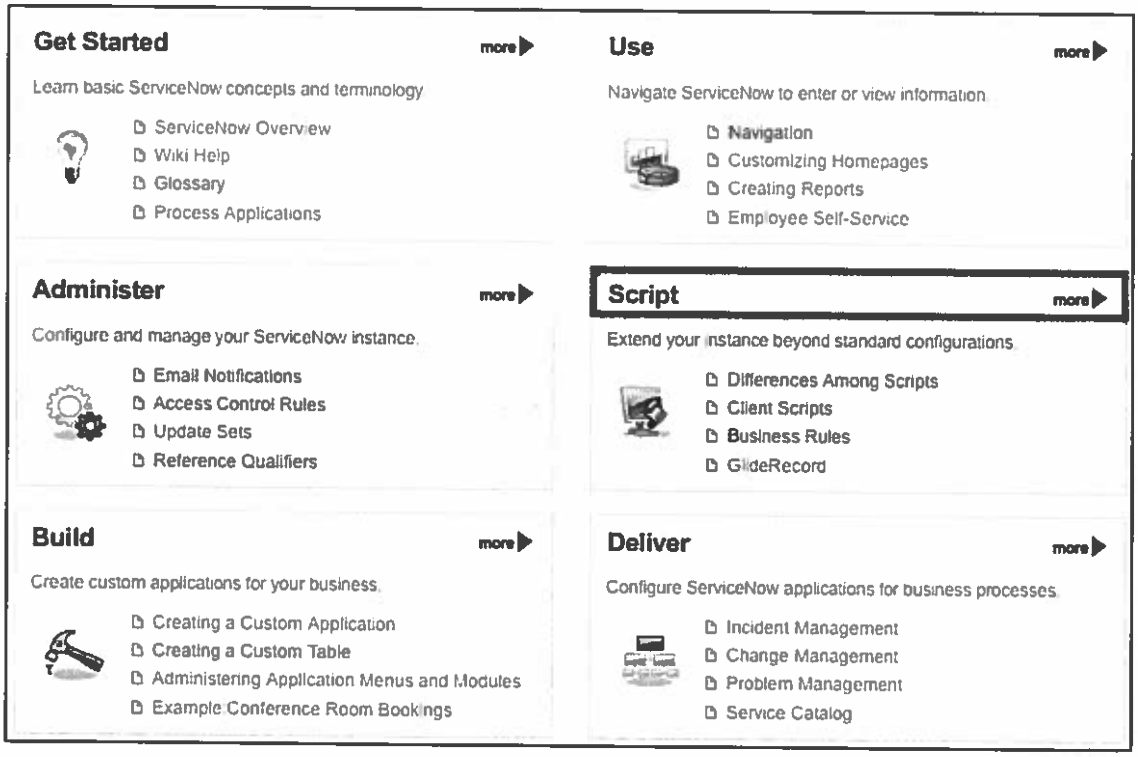
5. Search the web site for the string Ask Why.

   a. Locate the posting created by ctomasi in Chuck Tomasi's blog.

   b. Open and read the article. Summarize the content in one sentence:

6. Open wiki.servicenow.com.

7. Select **Script more >**.



8. Use the wiki page **Differences Among Scripts** from the General Scripting section to answer the following questions:

   a. What is a Client Script?

   b. What are the two most important differences among scripts?

9. Use the ServiceNow wiki to download the Scripting Guide book in PDF format. (Hint: Enter Book: in the wiki search box.)

10. Search the ServiceNow wiki for instructions on how to create your own PDF book from the wiki pages.

Best Practices

servicenow

- Always run the Syntax Checker when scripting
- Customize lists to make finding scripts faster
- Use the Edge for efficient navigation
- Use Flyouts when testing in order to quickly return to editing a script
- The ServiceNow website is more than the wiki; make good use of the community
- Join at least one SNUG

SNUGs are managed by ServiceNow users just like you. If there isn't a SNUG in your area, you can create one!

Module Recap

servicenow

Core Concepts

Use the Script Editor feature

Debug with the Syntax Checker

Personalize script lists

Customize the Edge

Know where to get scripting help

Real World Use Cases

Why you would use this

When you would use this

How often you would use this

Discuss: Why, when and how often would you use the capabilities shown in this module?

Client Scripts

Module 03

Objectives

Define what it means to be a Client Script
Know when to use a Client Script
Write, test, and debug Client Scripts
Use the g_form and g_user objects and methods
Retrieve reference records from the database
Write Client Scripts for Mobile
Compare and revert to script versions

Labs

3.1 Two Simple Client Scripts
3.2 g_form and g_user
3.3 Debugging Client Scripts
3.4 Client Scripting for Mobile
3.5 Client Scripting with Reference Objects
3.6 Script Versions

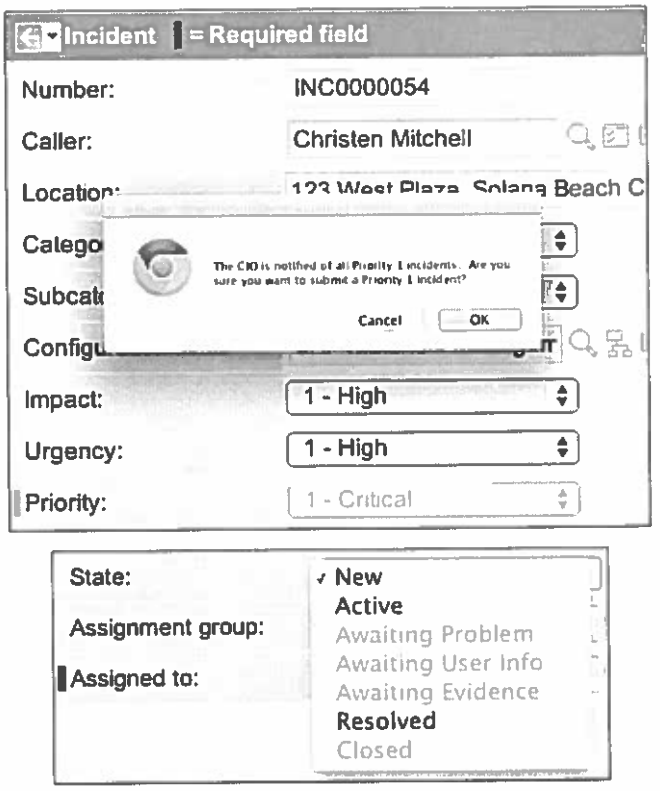
servicenow

In this module you will write, test, and debug Client Scripts.

What is a Client Script?

servicenow

- ServiceNow Client Scripts execute in a browser
- Client Scripts manage forms and fields in real time:
  - Modify choice list options
  - Hide/Show form sections
  - Display an alert
  - Set one field in response to another in real time
  - Make fields mandatory
  - Hide fields

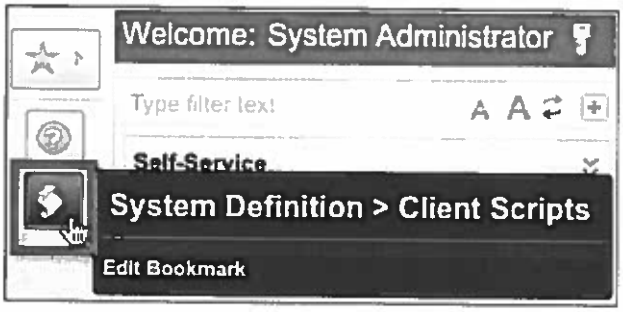


Although modern browsers largely interpret JavaScript the same way, you may still observe browser-dependent behaviors in client side scripts. For example, if a choice list has a restricted set of choices, some browsers will still display all of the choice list options but some will be grayed out. Other browsers will not display choice list options that have been removed.

Opening the Client Scripts List

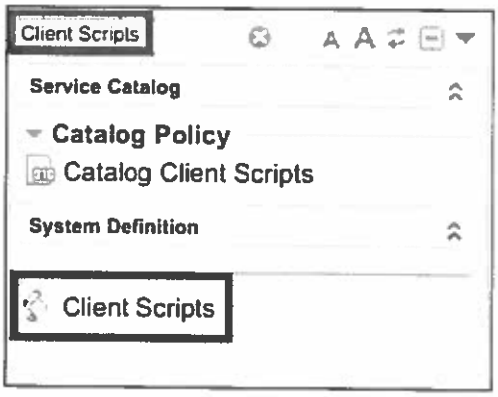
servicenow

1. Select Client Scripts from the Edge



OR

1. Enter **Client Scripts** in the Type Filter text field.
2. Select **System Definition > Client Scripts**



Another option is to use right-click on a form header and select **Personalize > Client Scripts** to see the Clients Scripts for that table.

What Exists Baseline?

servicenow

- Approximately 226 Client Scripts exist baseline
- Not all Client Scripts are Active in baseline
- Samples included as starting points

Name	Active	Updated	Table	View	Type
Sample: Setting an onBlur Handler	false	2011-08-24 07:18:14	Incident [incident]		onLoad
Sample:Priority One impact > 1	false	2011-08-24 07:17:53	Incident [incident]		onSubmit
Sample:Warn on Priority One	false	2011-08-24 07:18:40	Incident [incident]		onSubmit
Save with form button	true	2008-09-28 16:27:15	UI Action [sys_ui_action]		onChange
Schedule Item Days of Week	true	2010-11-08 06:13:26	Schedule Entry [cmn_schedule_span]		onChange
Schedule Item Float Day	true	2011-12-29 16:15:54	Schedule Entry [cmn_schedule_span]		onChange
Schedule Item Float Week	true	2011-12-29 16:15:47	Schedule Entry [cmn_schedule_span]		onChange
Schedule Item Month	true	2011-12-29 16:15:39	Schedule Entry [cmn_schedule_span]		onChange
Schedule Item Monthly Type	true	2010-11-08 06:13:37	Schedule Entry [cmn_schedule_span]		onChange
Schedule Item onLoad	true	2011-12-29 16:10:00	Schedule Entry [cmn_schedule_span]		onLoad
Schedule Item Repeat Count	true	2010-11-08 06:14:00	Schedule Entry [cmn_schedule_span]		onChange
Schedule Item Repeat Type	true	2010-11-08 03:46:59	Schedule Entry [cmn_schedule_span]		onChange
Schedule Item Start Date Time	true	2010-11-08 06:14:27	Schedule Entry [cmn_schedule_span]		onChange

To open the Client Scripts list:

1. Enter **Client Scripts** in the Type Filter text field.
2. Select **System Definition > Client Scripts.**

OR

Open the Client Scripts list from the Favorites list on the Edge.

Be cautious about modifying baseline scripts. Save a copy and modify the copy rather than the original.

OR

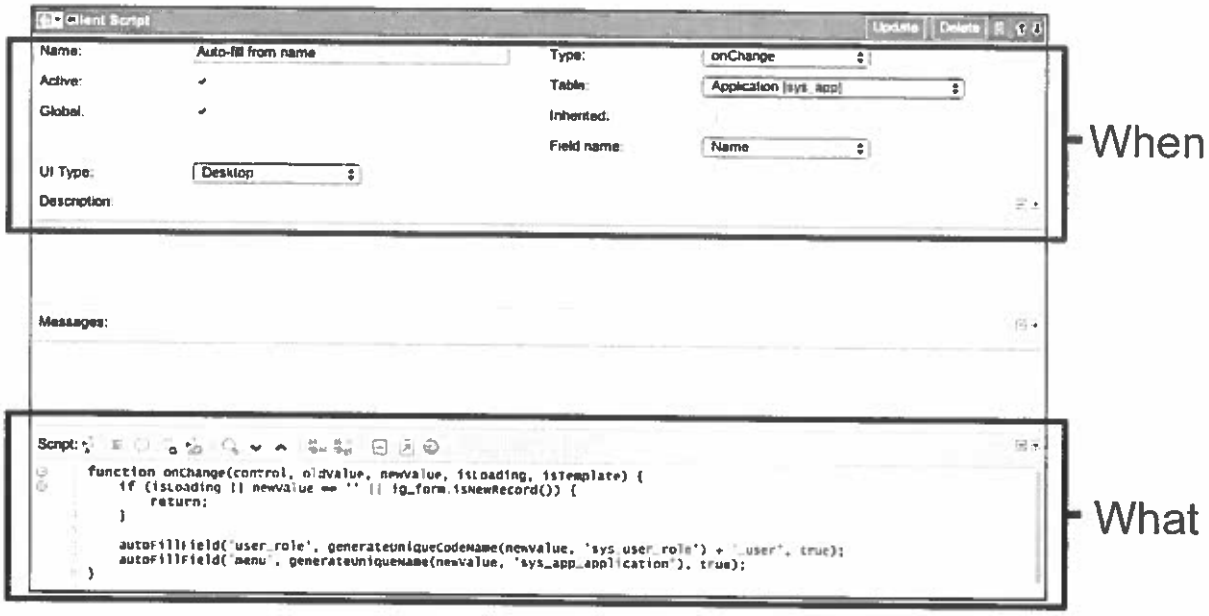
From a form or list use the context menu and select **Personalize >Client Scripts** to see the clients scripts for that table.

If you modify a baseline script in any way, that script will not be updated in future ServiceNow updates. Once a baseline script is modified, you own it!

When vs. What

servicenow

- Trigger specifies *when* to execute
- Script specifies *what* to do



All scripts have a trigger that specifies when a script's logic should execute. The trigger configuration fields depend on the script type.

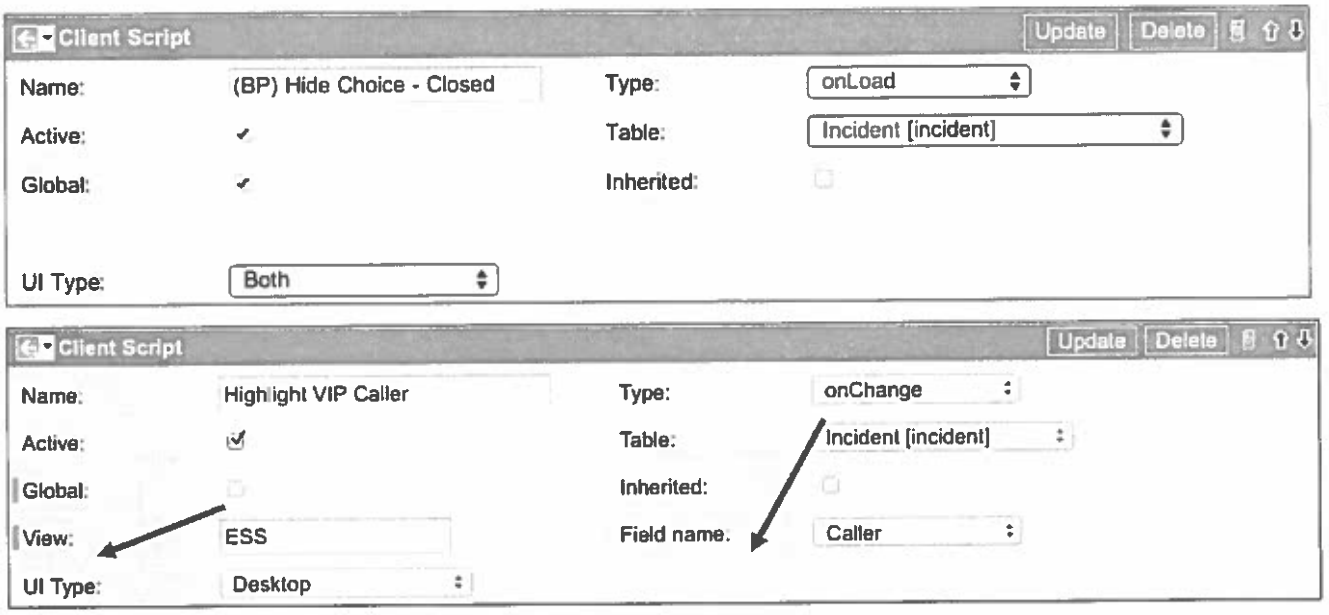
The Description field is for documenting the script. Include information like who wrote the script, what business requirement the script is for and any other pertinent information.

The Messages field is used for internationalizing output to the user. For example, if the script creates an alert that says Hello World, the string "Hello World" would appear in the Messages field on its own line. If an entry exists in the sys_ui_message table with that same key but a localized language, the localized language version is presented to the user even though the script uses the version from the Messages field.

Client Script Trigger

servicenow

- Trigger condition must be met for the Client Script to execute

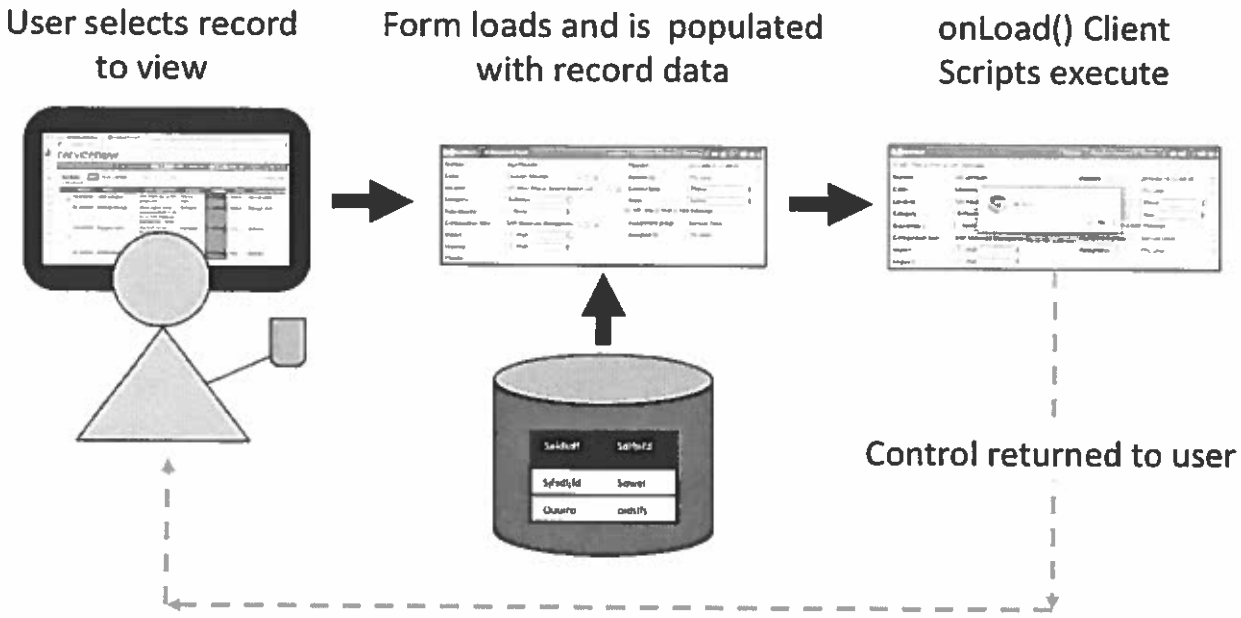


While on the Client Scripts list, select **New** to create a new Client Script.
Name: Name of Client Script. Use a standard naming scheme to identify custom scripts.
Active: If selected the script is executing in the runtime environment.
Global: If Global is selected the script applies to all Views. If the Global field is not selected you must specify the View.
View: Specifies the View to which the script applies. The View field is only visible when Global is not selected. A script can only act on fields that are part of the selected form View. If the View field is blank the script applies to the Default view.
UI Type: Select whether the script executes for Desktop and Tablet or Mobile or both.
Type: Select when the script runs: onChange, onLoad, onSubmit, or onCellEdit.
Table: Form to which the script applies.π
Inherited: Execute the script for forms from any extended tables when selected.
Field Name: Used only if the script responds to a field value change (onChange or onCellEdit); name of the field to which the script applies.

onLoad()

servicenow

- Script runs when a form meeting the trigger condition loads and before control is given to the user

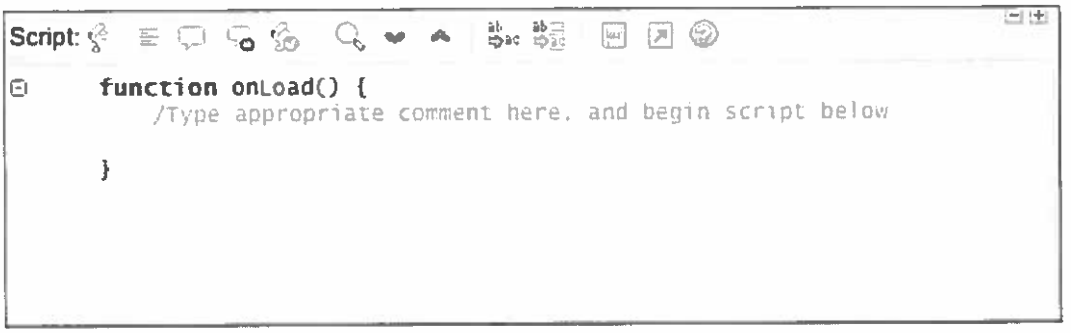


While onLoad() Client Scripts executes, users have no ability to modify form fields.

onLoad()

servicenow

- Typically used to manipulate a form's appearance or content
- The onLoad() function template is automatically inserted in the Script field

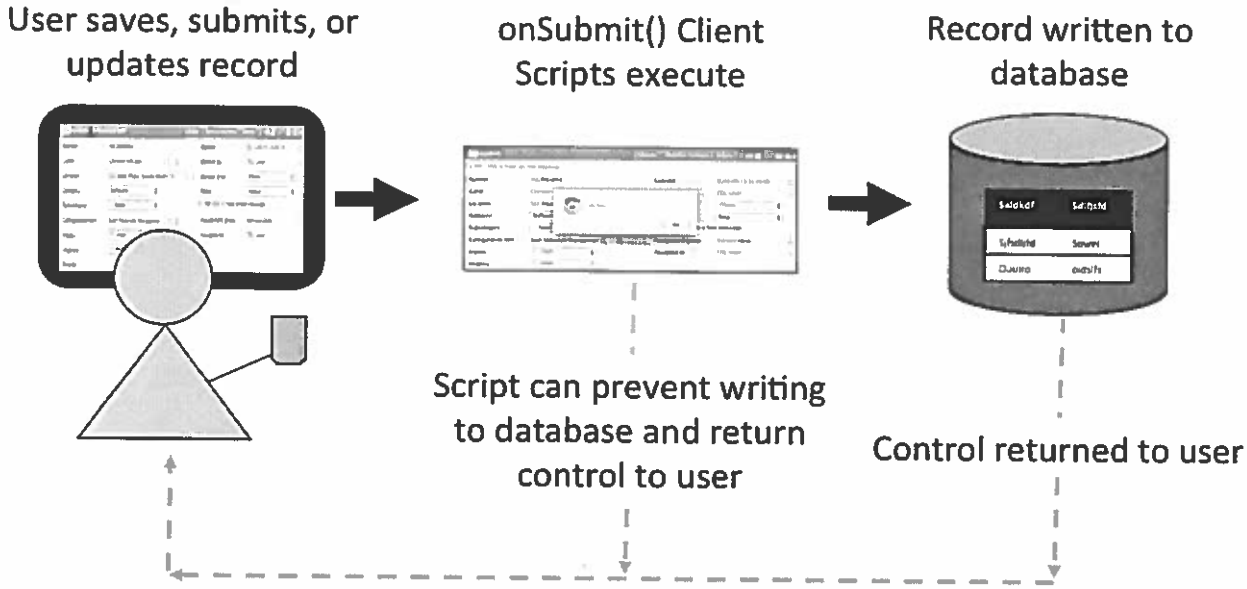


The onLoad() function has no arguments passed to it.

onSubmit()

servicenow

- Script runs when a form meeting the trigger condition is saved, updated, or submitted

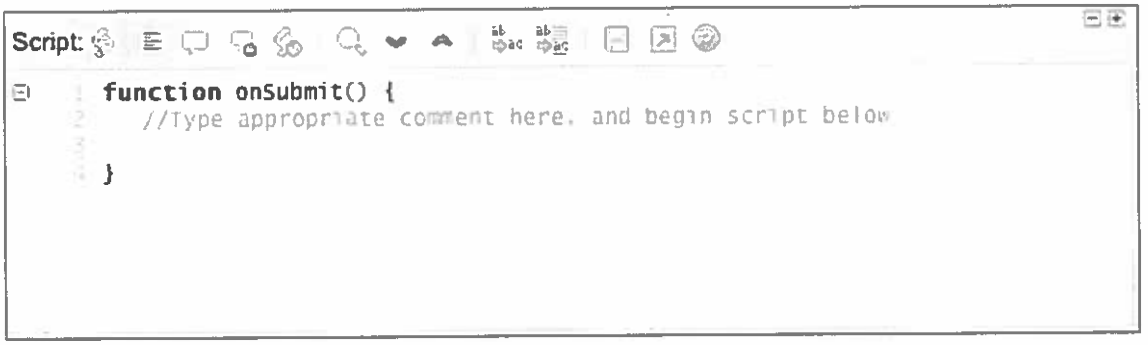


Users have no access to a form's fields while onSubmit() Client Scripts execute.

onSubmit()

servicenow

- Typically used for field validation
- The onSubmit() function template is automatically inserted in the Script field



The onSubmit() function has no arguments passed to it.

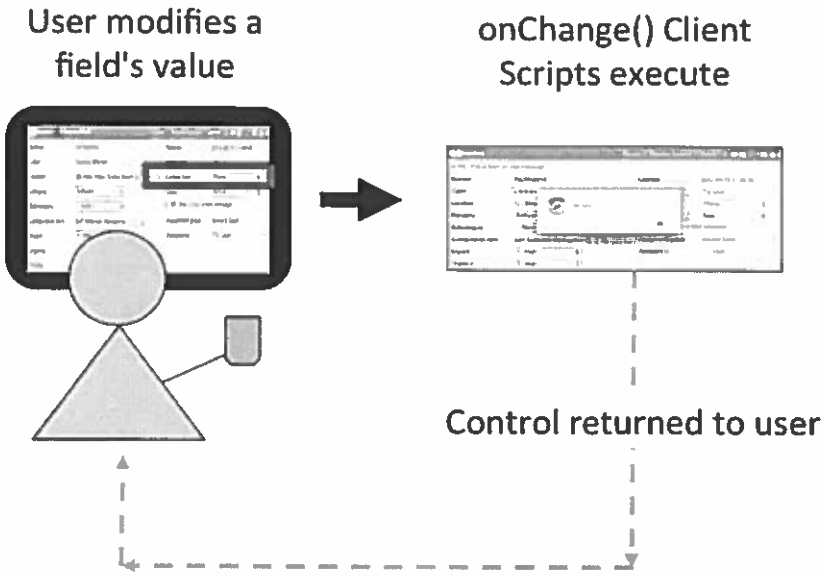
Tip: You can cancel a form submission by returning false.

```
function onSumbit(){
  if(!myCondition){
     return false;
   }
  else {
    //perform some logic
  }
}
```

onChange()

servicenow

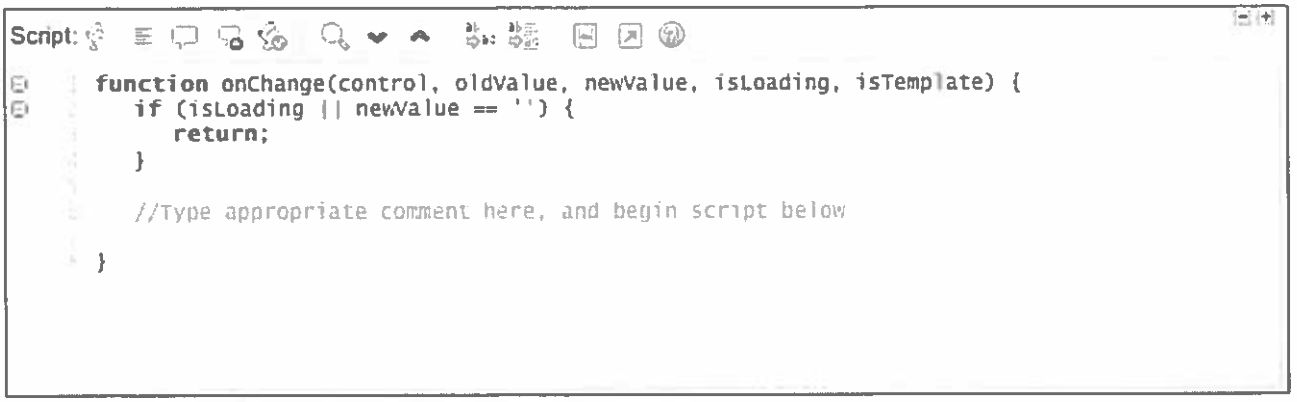
- Script runs when a particular field's value on a form changes



onChange()

servicenow

- Typically used to respond to field values of interest or modify one field's value in response to another
- The onChange() function template is automatically inserted in the Script field



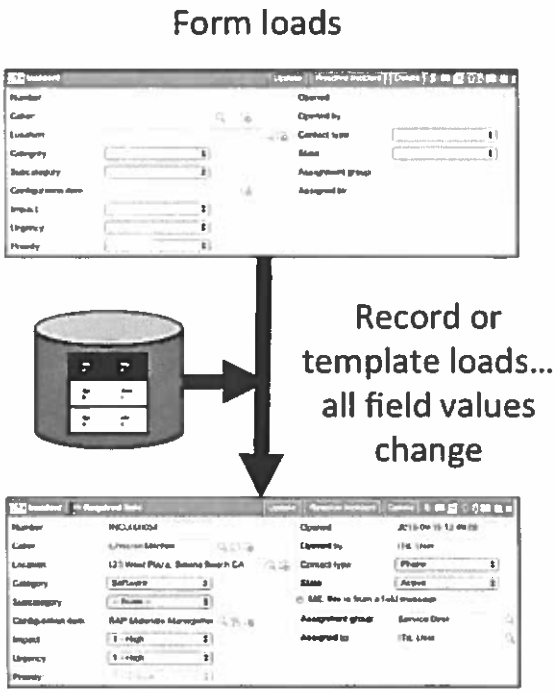
The onChange() function is automatically passed 5 parameters:

1. control: name of field whose value changed. This is the value of the Field Name from the trigger condition.
2. oldValue: value of the control field when the form loaded and prior to the change. For example, if the value of Assigned To changes from Chuck to Mary Ellen the value of the parameter oldValue is Chuck. oldValue will always be Chuck (the value when the form loaded) no matter how many times the Assigned To value changes after the original form load.
3. newValue: value of the control field after the change. For example, if the value of Assigned To changes from Chuck to Mary Ellen the value of the parameter newValue is Mary Ellen.
4. isLoading: boolean value indicating whether the change is occurring as part of a form load. Value is true if change is due to a form load. A form load means that all of a form's fields changed.
5. isTemplate: boolean value indicating whether the change occurred due to population of the field by a template. Value is true if change is due to population from a template.

onChange() Template if Statement

servicenow

- When a form loads, ALL field values change
  - onChange() template if statement by default prevents execution when a field's value changes due to a form load
- When a template loads, field values change
  - onChange() template if statement does not prevent execution when a field's value changes due to a template



The template's if statement aborts script execution if a field's value changed due to a form load or if newValue has no value.

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

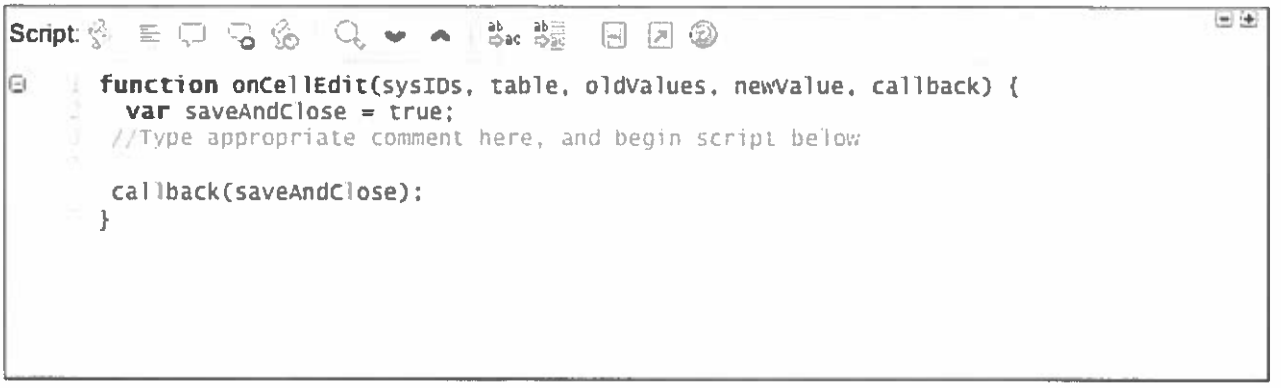
Modify the script logic to change this behavior if necessary. For example, you might also check to see if the field value change was due to a template load.

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

onCellEdit()

servicenow

- Script runs when a particular field value on a list changes
- Can be applied to multiple records
- The onCellEdit() function template is automatically inserted in the script field



The onCellEdit() function is automatically passed 5 parameters:

1. **sysIDs – sys_id of the edited item(s).**
2. **table – the table name of the edited item(s).**
3. **oldValues – the old value of the edited cell(s).**
4. **newValue – the new value of the edited cell(s). Is the same for all edited items.**
5. **callback– A callback that will continue the execution of any other related cell edit scripts. If 'true' is passed as a parameter, then the other scripts are executed or the change is committed if there are no more scripts. If 'false' is passed as a parameter, then any further scripts are not executed and the change is not committed.**

When using this function, you are required to pass back either true or false in the callback function.

Lab

servicenow

3.1 Two Simple Client Scripts

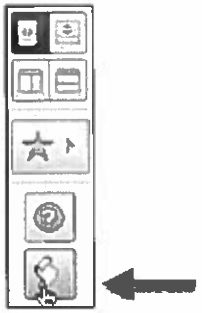
Lab Goal

In this lab you will write and test two simple Client Scripts. The first script is an onLoad() Client Script which generates an alert. The second script is an onChange() Client Script to take two different actions based on how the oldValue compares to the newValue.

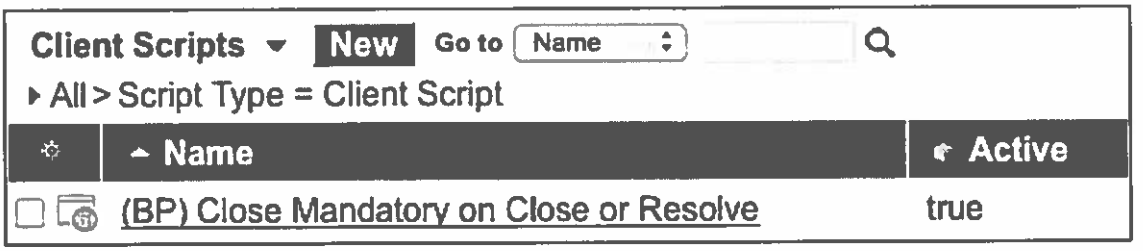
Lab 3.1
Two Simple Client Scripts

onLoad() Client Script

1. Open the Client Scripts list from the Edge.

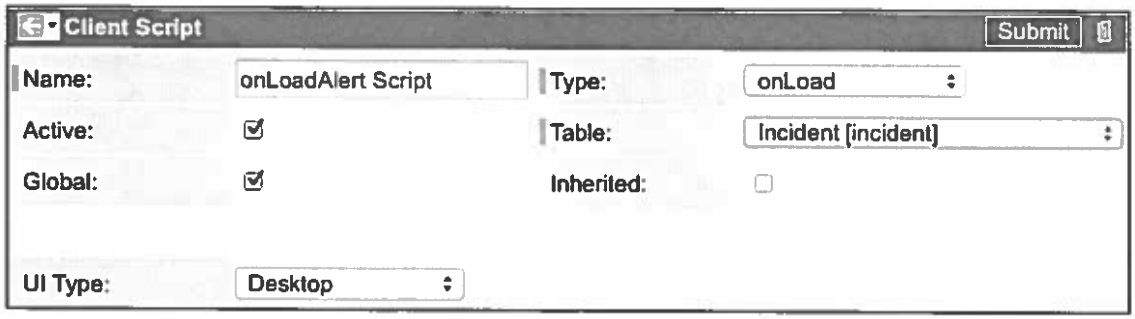


2. Create a new Client Script by selecting the **New** button.

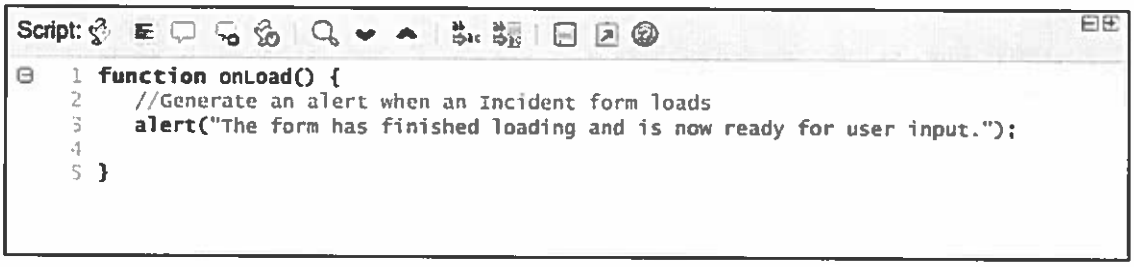


3. Configure the Client Script trigger:

   Name: **onLoad Alert Script**
   Active: **Selected (checked)**
   Global: **Selected (checked)**
   UI Type: **Desktop**
   Type: **onLoad**
   Table: **Incident [incident]**
   Inherited: **Not selected (not checked)**



4. Note that the onLoad() function template populated the script field. Add a comment and an alert to the onLoad() function.



5. If real time syntax checking is disabled, use the Syntax Checker () to check for syntax errors. Correct any problems.

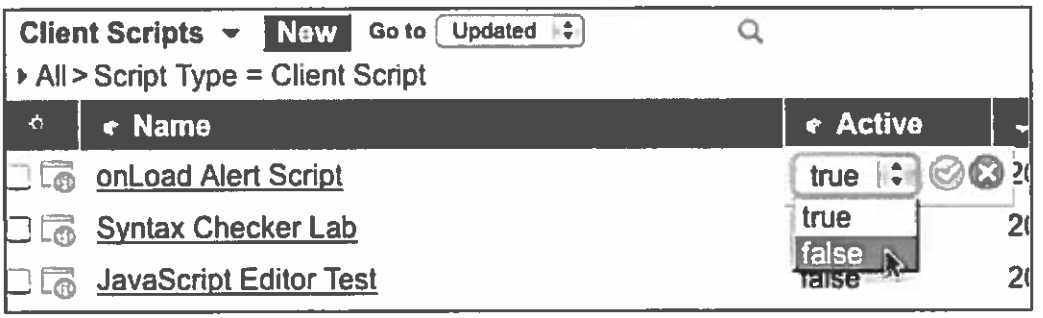
6. Save the script by selecting the **Submit** button. If you have saved the script previously select the **Update** button.

7. Open the list of Open Incidents from the Edge.

8. Open the Incident of your choice. What happens when the form loads? Is this the behavior you expected?

9. Create a new Incident. What happens when the form loads for the new incident? Is this the behavior you expected?

10. Open the Client Script list view and set the Active value of the onLoad Alert Script to false.



onChange() Client Script

1. Create a new Client Script.

2. Configure the Client Script trigger:

   Name: **onChange Alert Script**
   Active: **Selected (checked)**
   Global: **Selected (checked)**
   UI Type: **Desktop**
   Type: **onChange**
   Table: **Incident [incident]**
   Inherited: **Not selected (not checked)**
   Field Name: **Impact**

3. Add logic to the script to generate two different alerts:

   1) If the new impact is greater than the old impact generate an alert that says the incident impact has been escalated.

   2) If the old impact is greater than the new impact generate an alert that says the incident impact has decreased.

4. Use the real time syntax checking or the Syntax Checker to check for syntax errors.

5. After submitting your script, test it by changing the impact on an open incident. Did an alert appear? Is it the alert you expected? Troubleshoot any issues.

6. Make the onChange Alert Script inactive.

What data can I use in a Client Script?

servicenow

- Local variables declared in script
- Client Script Global Variables
  - –g_form
  - –g_user
  - –g_scratchpad

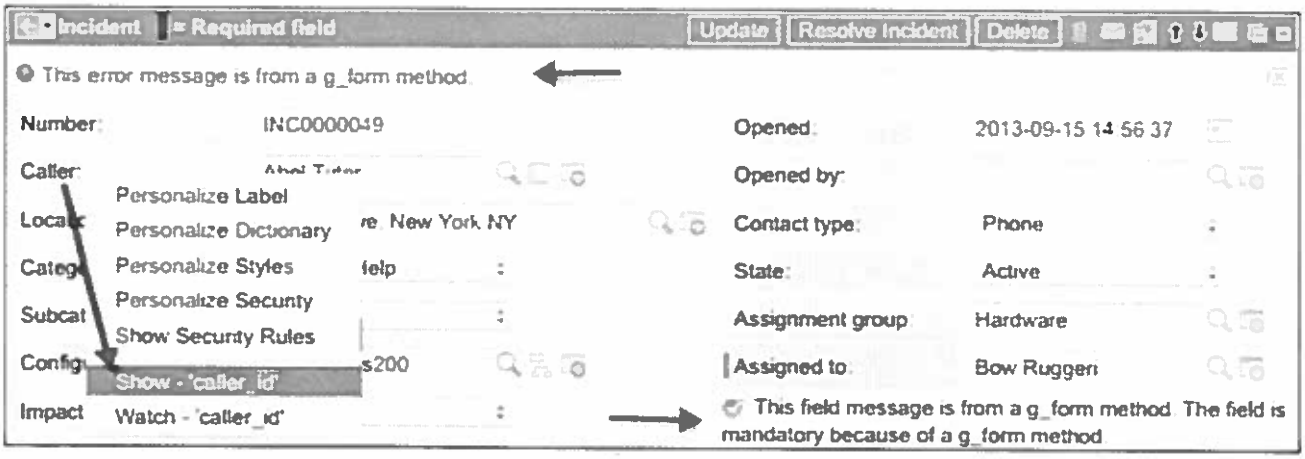
ServiceNow global variables:

- g_form is an object whose properties are methods used to manage form fields.
- g_user is an object whose properties contain session information about the currently logged in user and their role(s).
- g_scratchpad is a global object passed to a Client Script from a server side script known as a Display Business Rule. The object's properties and values are determined by the server side script.

g_form Object

servicenow

- g_form is an object whose methods are used to manage form fields and their values
- Methods require use of field names and not labels



The g_form object methods refer to fields by their field names and not by their labels. To see the field name, right-click the field's label. The field name appears at the bottom of the right-click menu.

Client Scripts

g_form Object Methods

servicenow

- Method categories
  - Display Settings: flash(), showFieldMsg()
  - Field Information: getValue(), getReference()
  - Change Field: setValue(), clearValue()
  - Change Choice List: addOption(), clearOptions()
  - Form Information: getSections(), isNewRecord()
  - Form Action: addInfoMessage(), enableAttachments()
- Complete list of g_form methods and their syntax available on the ServiceNow wiki

g_form.flash() – flashes a field to draw attention to it
g_form.showFieldMsg() – displays a message under a form field
g_form.getValue() – retrieves a field's value
g_form.getReference() – retrieves a reference object from the database
g_form.setValue() – sets a field's value
g_form.clearValue() – clears a field's value
g_form.addOption() – adds an option to a Choice List
g_form.clearOptions() – removes all options from a Choice List
g_form.getSections() – returns the elements of a form's section as an array
g_form.isNewRecord() – returns true if a record has never been saved
g_form.addInfoMessage() – displays an informational message at the top of a form
g_form.enableAttachments() – allow attachments to be added to a record

g_form Method Syntax - wiki

servicenow

- Method Summary

Return Value	Details
void	**flash**(widgetName, color, count) Flashes the specified color the specified number of times. Used to draw attention to a particular field.

- Detailed Description

void flash(widgetName, color, count)

Flashes the specified color the specified number of times in the field. Used to draw attention to a particular field.

Parameters:

widgetName - specifies the field with <table name>.<fieldname>.

color - RGB color or acceptable CSS color like "blue" or "tomato."

count - integer that determines how long the label will flash.

- use 2 for a 1-second flash
- use 0 for a 2-second flash
- use -2 for a 3-second flash
- use -4 for a 4-second flash

Returns:

void

Example:

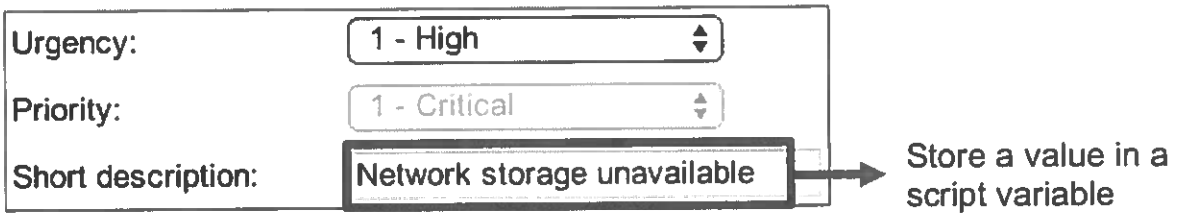
```
g_form.flash("incident.number", "#FFFACD", 0);
```

The ServiceNow wiki provides a table of g_form methods as a quick reference. Click the method name to view the detailed syntax description. Most methods have the Parameters and Return value documented. Some methods also contain example code.

g_form.getValue()

servicenow

- Retrieves a fields value from a form (not the database)



- Method syntax:

```
var <varName> = g_form.getValue('<field_name>');
```

- You must pay attention to a field's data type when using this method

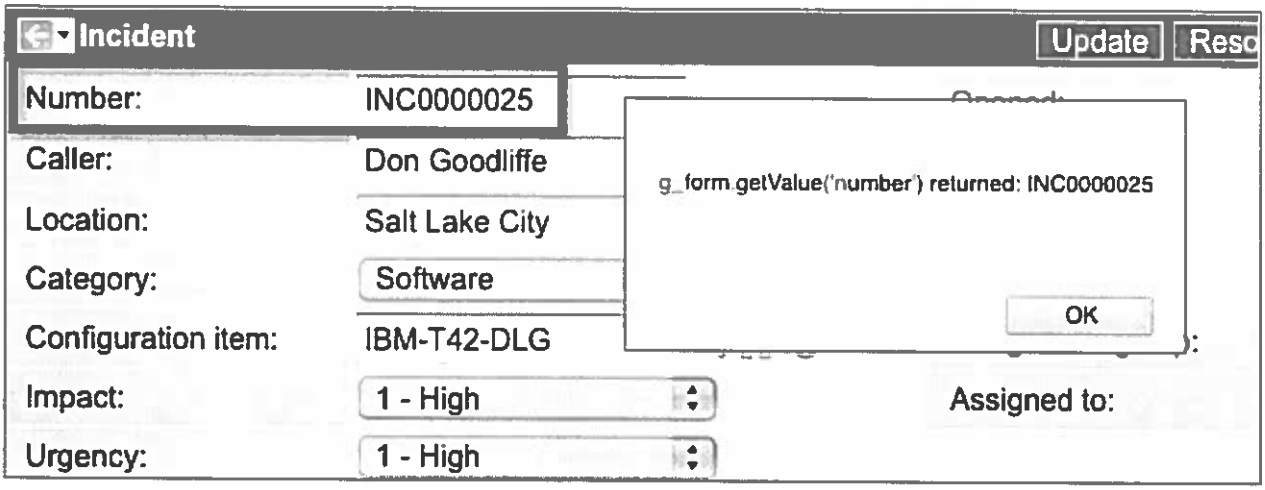
This method cannot retrieve values from the database; the field must exist on the form. The g_form.getValue() method can retrieve values from forms even if the field is not visible in the current view or to the logged in user.

g_form.getValue() is a commonly used g_form method. Since JavaScript is weakly typed, it isn't always necessary to verify data type when scripting. In the case of the g_form.getValue() method, you must pay attention to data type or your script may have unexpected results. The g_form.getValue() method always returns a string despite the data type of the field. If returning a number is important, use the g_form.getIntValue() or g_form.getDecimalValue() methods instead.

g_form.getValue() - Text Field

servicenow

- g_form.getValue('number')
  - Returns the contents of the Number (number) text field

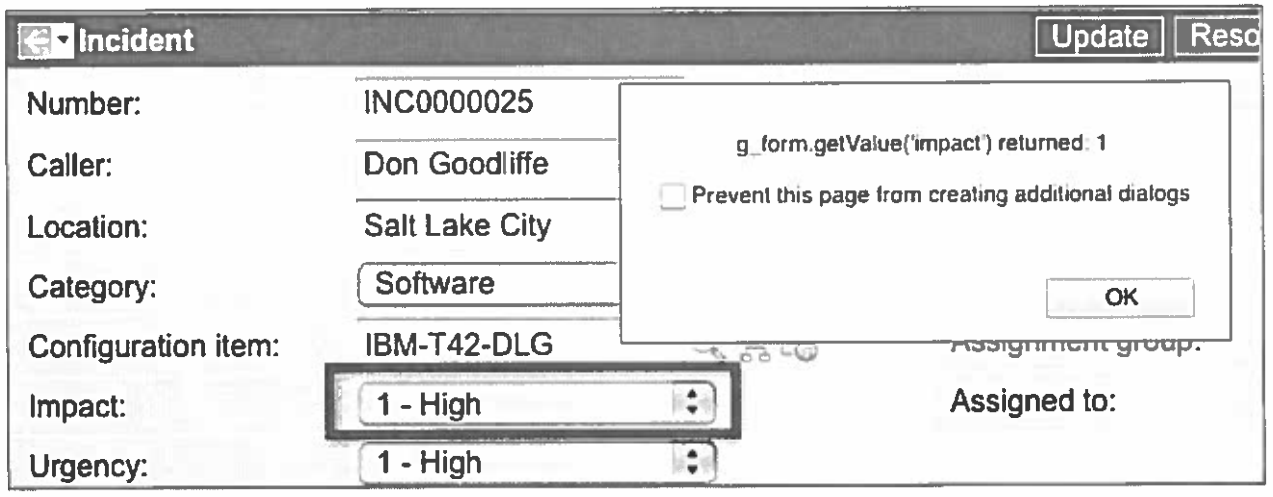


When g_form.getValue('<field name>') is passed a field of type string, the method returns the contents of the text field exactly as it appears on a form.

g_form.getValue() – Choice List

servicenow

- g_form.getValue('priority')
  - Returns the value of the Priority (priority) choice list selection



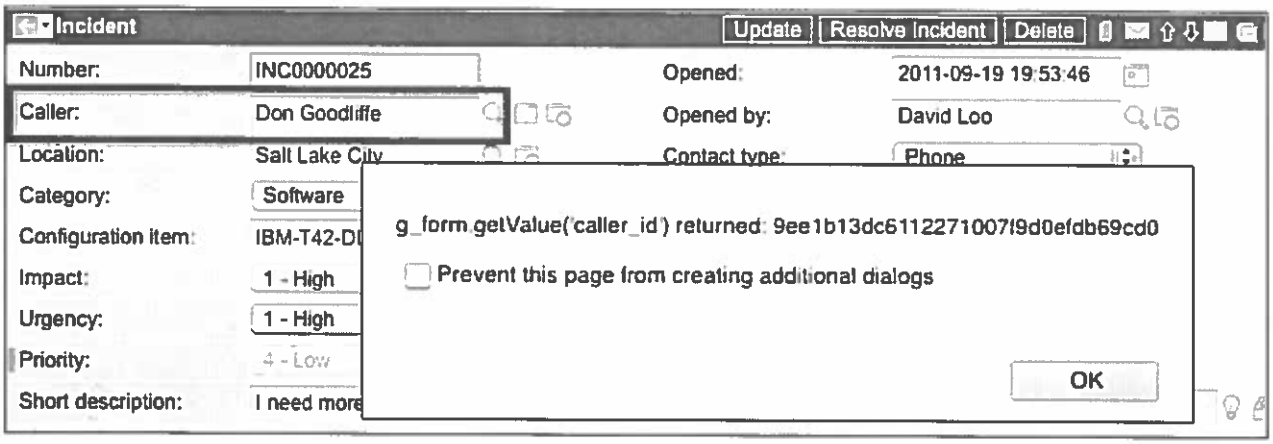
When g_form.getValue('<field name>') is used with a choice list field, the method returns the *value* of the choice list and not the label.

To see choice list options right-click the field name and select Show Choice List.

g_form.getValue() – Reference Field

servicenow

- g_form.getValue('caller_id')
  - –Returns the sys_id of the Caller (caller_id)reference field



When g_form.getValue('<field name>') is used with a reference field, the method returns the sys_id of the referenced record.

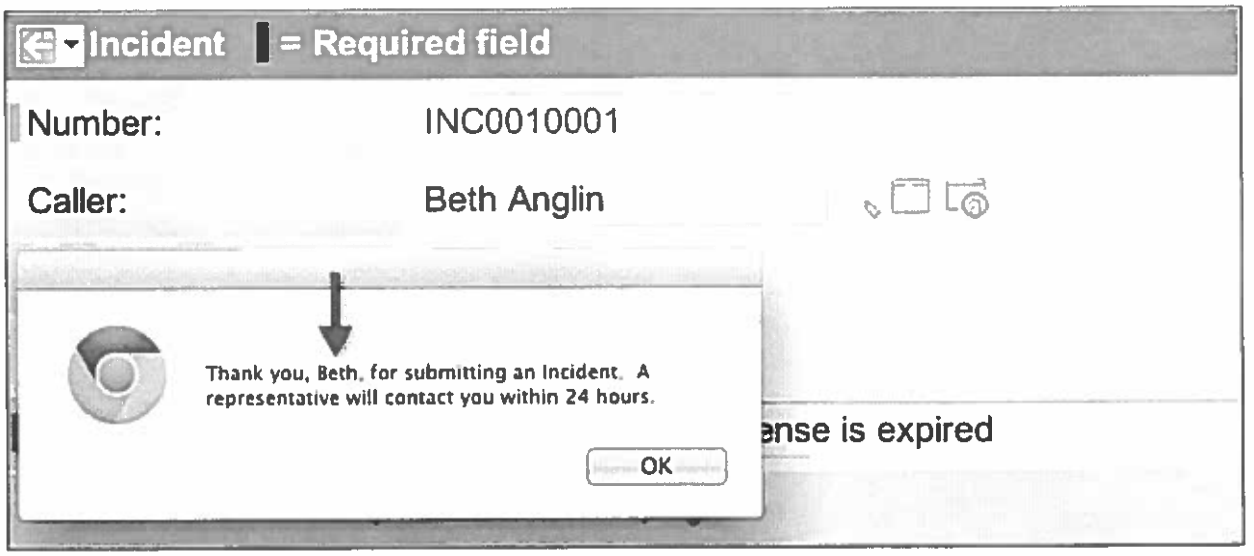
A sys_id is the database's unique key for a record.

Client Scripts

g_user Object

servicenow

- g_user is an object containing information about the currently logged in user
- Primarily used for personalization and role verification



For the form shown, the g_user object contains the following properties and values. The example shown is for the user Joe Employee.

```
g_user = {
        userName: "employee",
        userID: "681ccaf9c0a8016400b98a06818d57c7",
        firstName: "Joe",
        lastName: "Employee"
  }
```

g_user Object Properties

servicenow

- Non-method properties
  - userName
  - userID
  - firstName
  - lastName
- Methods
  - getFullName()
  - hasRole()
  - hasRoleExactly()
  - hasRoleFromList()
  - hasRoles()
- Partial g_user object for Joe Employee

```
g_user = {
        userName: "employee",
        userID: "681ccaf9c0a8016400b98a06818d57c7",
        firstName: "Joe",
        lastName: "Employee"
    }
```


See the ServiceNow wiki for a complete list of g_user methods and their syntax.

g_user.getFullName() – returns the logged in user's first name and last name separated by a space
g_user.hasRole() – returns true if the logged in user has the role passed in the method call or has the admin role
g_user.hasRoleExactly() – returns true if the logged in user has the role passed in the method call
g_user.hasRoleFromList() – returns true if the logged in user has at least one role from the passed in list or has the admin role
g_user.hasRoles() – returns true if the logged in user has any role

DO NOT rely on the g_user methods to apply security. Client side security is easily defeated using developer tools built into browsers. Access Control or another server side security strategy is recommended.

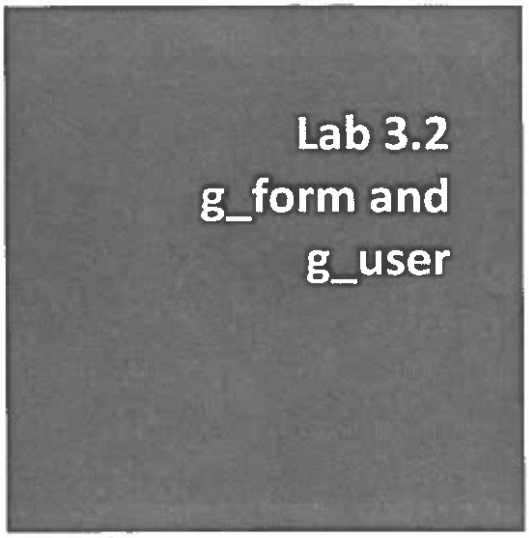
Lab

servicenow

3.2 g_form and g_user

Lab Goal

In this lab you will practice writing a Client Script using the g_form and g_user objects and their methods.



Using g_form and g_user

1. Create a new Client Script.

2. Configure the Client Script trigger.

   Name: **Client Script Objects**
   Active: **Selected** (checked)
   Global: **Selected** (checked)
   UI Type: **Desktop**
   Type: **onSubmit**
   Table: **Incident [incident]**
   Inherited: **Not selected** (not checked)

3. Examine the pseudo-code for the script you will write:

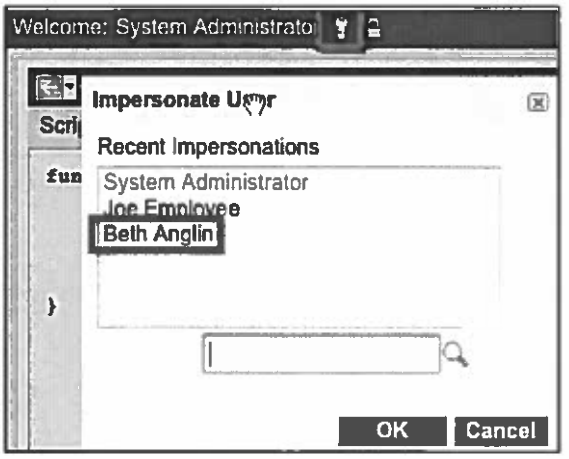
```
When the form is submitted, saved, or updated.
    Create the ans variable with no value.
    If Impact and Urgency are both high and the user does not have the admin role
        Display a confirmation box asking if the user really wants to submit
        a Critical priority incident.
            If the user cancels the submission
                Generate an alert stating that the incident was not
                submitted.
                Add a field message below the Urgency field.
        Return true or false based on the confirmation box response.
```

4. Write the script:

```
function onSubmit() {
  var ans = '';
  if(g_form.getValue('impact') == 1 && g_form.getValue('urgency') == 1
  && !g_user.hasRole('admin')){
    ans = confirm('The CIO is notified of all Priority 1 incidents.  Do
    you want to submit this proiority 1 incident?');
  }
  if(!ans){
    alert('Incident not submitted.');
    g_form.showFieldMsg('urgency','If Urgency and Impact are both High
    the Priority will be critical.','info');
  }
  return ans;
}
```

5. You cannot completely test this script as the admin user. Why not?

6. Impersonate Beth Anglin:



7. Create an incident and set both the Impact and Urgency values to High. Note that Priority is automatically set to Critical.

8. Scroll down the incident form until the Impact and Urgency fields are off the screen.

9. Save (not Submit) the incident to stay on the same form page by right clicking the Closure information bar or Related Records bar and selecting Save.

10. When the confirmation window opens, **Cancel** the submission.

11. What happened on the incident form when you cancelled the submission? Are the same fields visible as when you cancelled the submission? Why did this happen?

12. Test again but this time select **OK** when prompted by the confirmation dialog box. Did you get the results you expected?

13. Modify the confirmation dialog box so the text includes the logged in user's first name. Which property do you need to use?

14. Test the modified script.

15. Make the Client Script Objects script inactive.

Debugging Client Scripts - Desktop

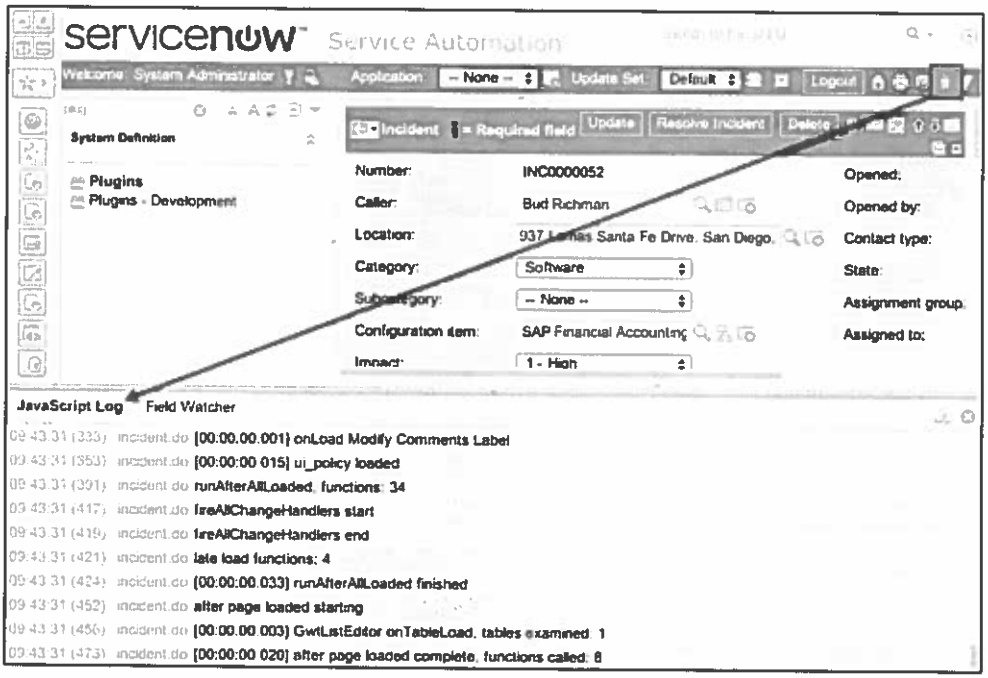
servicenow

- ServiceNow Tools
  - –JavaScript Log
  - –Field Watcher
  - –Response Time Indicator
- JavaScript Tools
  - –Try/Catch

JavaScript Log

servicenow

- Administrators can open the JavaScript Log



Follow these steps to open the JavaScript Log:

1. Select the "bug" icon (admin users only). The JavaScript Log opens in a tab in a new frame.
2. Create the conditions necessary to test your script.
3. View jslog() messages in the JavaScript Log.

The JavaScript Log must remain open in order to debug. If the JavaScript Log is closed, select the "bug" icon to open it again.

The JavaScript Log is only available in Desktop mode; it is not available in Tablet or Mobile mode.

jslog()

servicenow

- The jslog() method writes to the JavaScript Log

Script:

```
function onLoad() {
    var x = 7;
    jslog("This message is from jslog().");
    jslog('The value of x = ' + x);
    jslog('The incident state is: ' + g_form.getValue('state'));
    jslog('Welcome ' + g_user.getFullName() + '. \nIt is good to see you here.');
}
```

JavaScript Log Field Watcher

```
09:43:31 (331)  incident.do This message is from jslog().
09.43:31 (331)  incident.do The value of x = 7
09:43:31 (331)  incident.do The incident state is: 1
09:43:31 (332)  incident.do Welcome System Administrator. It is good to see you here.
09:43:31 (332)  incident.do [00:00:00.001] onLoad ME My First Client Script
09:43:31 (332)  incident.do [00:00:00.000] onLoad onLoadAlert Script
```

Any argument passed to the method appears in the JavaScript Debug window. The arguments can be:

- Strings
- g_form properties or methods
- g_user properties or methods
- Variables
- JavaScript string escape characters such as \n (new line) and \t (tab) (acceptable but ignored)

This strategy allows you to write any debugging information you want and is not restricted to field values. The JavaScript Log appears only for the admin user who opened it; no other users are impacted.

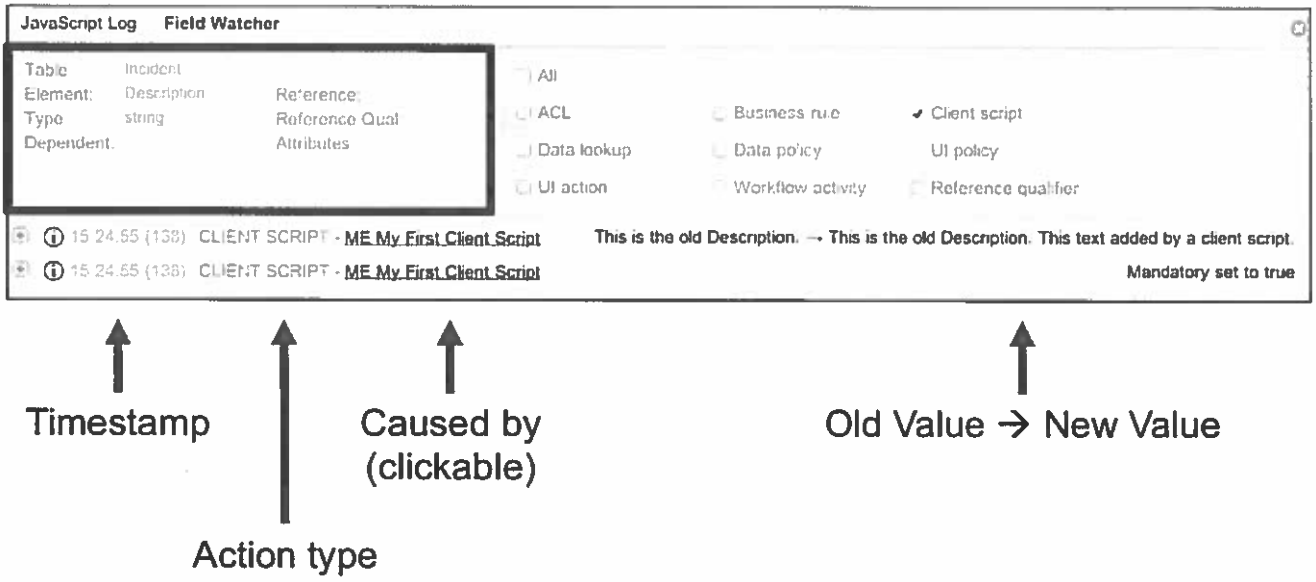
If multiple users are logged in and editing scripts, the JavaScript Log can contain a lot of data. Prepend your jslog() messages with an easily identifiable string such as the script name and/or your name so you can quickly search through the debug messages.

jslog() method calls do not need to be removed for test and production instance. jslog() does not write to a table.

Field Watcher

servicenow

- The Field Watcher tracks and displays actions the system performs on a form field



Timestamp

Action type

Caused by (clickable)

Old Value → New Value

Client Scripts

The Field Watcher must remain open in order to debug. If the Field Watcher is closed, select the "bug" icon to open it again.

The Field Watcher is only available in Desktop mode; it is not available in Tablet or Mobile mode.

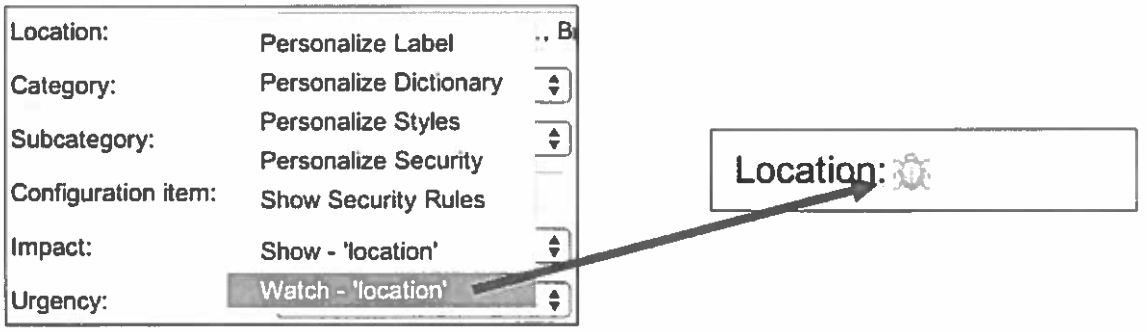
Although the Field Watcher can be configured to display field affecting information for many types of actions, this discussion focuses on the Client Script activity.

Select the Clear Log (eraser) icon in the upper right of the Field Watcher to clear the field activity information.

Watching a Field

servicenow

- Select a field to watch



- Can only watch one field at a time
- Fields must be visible on a form to be watched

Selecting a field to watch does not impact other users.

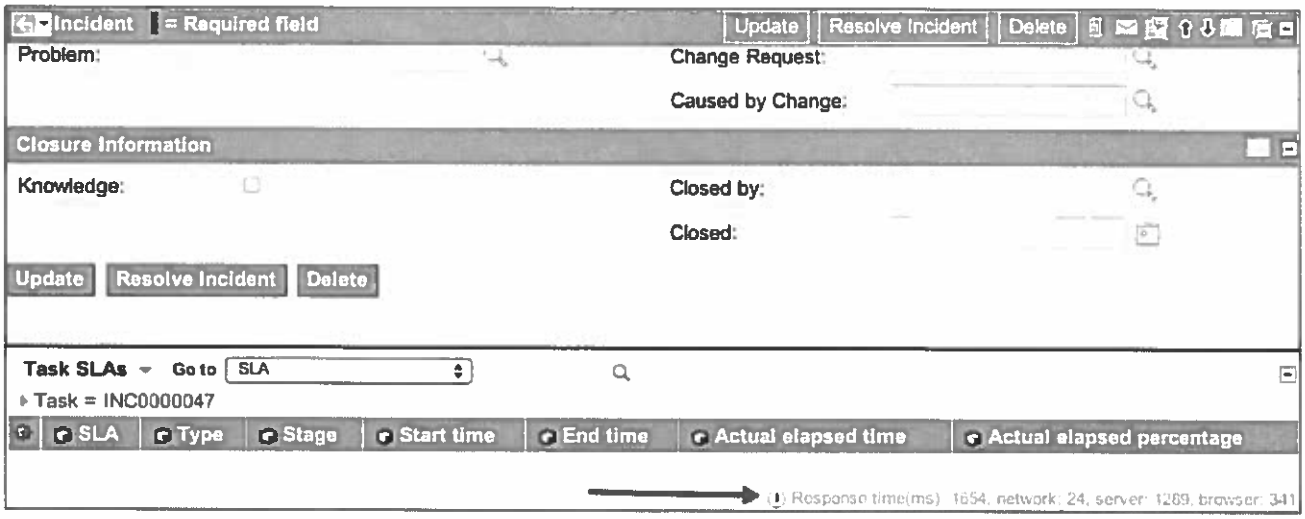
Use the jslog method to log debugging information for fields that are not visible on a form.

To stop watching a field, right-click on the field label and select Unwatch – 'field_name'.

Response Time Indicator

servicenow

- May appear at the bottom right of forms and lists
- Displays processing time for each step of a form load



Select the clock icon to toggle the Response Time Indicator on/off.

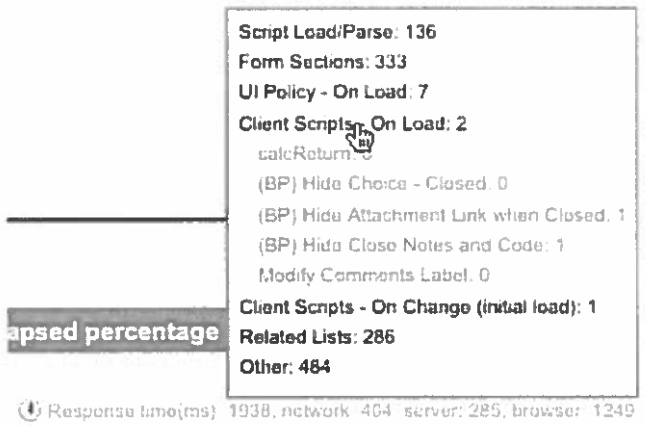
Use this strategy to look for script issue causing long load times.

Administrators can disable the response time indicator by setting the glide.ui.response_time property to false.

Response Time Indicator Results

servicenow

- Times are in milliseconds
  - –Total load time
  - –Network
  - –Browser (click the browser link for more details)
  - –Server



The Response Time Indicator is useful for locating the cause of slow page loads.

The category Other represents time spent on administrative overhead.

Try/Catch

servicenow

- This example passes the Syntax Check but has a runtime error

Script:

```
function onLoad() {

    try{
        helloworld();
    }
    catch(err){
        jslog('A JavaScript runtime error occurred: ' + err.message);
    }
}
```

JavaScript Log Field Watcher

```
11:38:24 (058)  incident.do  A JavaScript runtime error occurred: helloWorld is not defined
11:38:24 (058)  incident.do  [00:00:00.000] onLoad ME My First Client Script
11:38:24 (058)  incident.do  [00:00:00.000] onLoad onLoadAlert Script
```


Try/Catch is a JavaScript debugging strategy used to trap runtime errors.

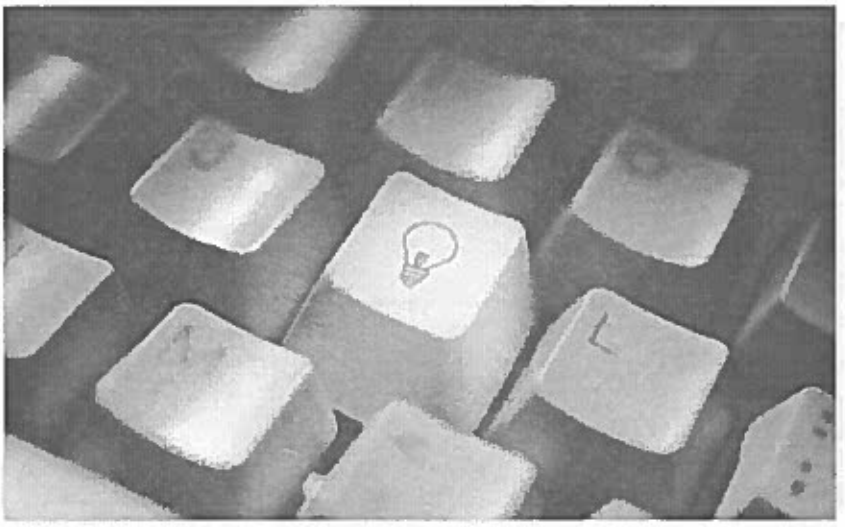
```
try{
    //code to execute goes here
  }
  catch(err){
    //code to deal with error here
  }
```

err is a JavaScript object with properties description, message, name, and number.

You can throw your own error messages for the catch function using the JavaScript throw() function. Try/catch only traps runtime errors. Using throw() you can also catch user errors such as entering an invalid data value in a field.

Lab

servicenow



3.3 Debugging Client Scripts

Lab Goal

In this lab you will practice debugging a Client Scriptsusing ServiceNow and JavaScript strategies.

Lab 3.3 Debugging Client Scripts

Preparation

1. Download the Debugging Client Scripts.xml file from the Knowledge Base on trainingdept.service-now.com. Log in as scripting / scripting.
2. On your class instance, select the lock icon () in the ServiceNow banner to give yourself elevated security privileges.
3. Import the Debugging Client Scripts.xml file.
    a. Open the Client Scripts list.
    b. Right-click on the list header and select **Import XML.**
    c. Select the **Browse** button and navigate to the downloaded file.
    d. Choose the file and select the **Open** button.
    e. Select the **Upload** button.

Procedure

1. Locate the Debugging Client Scripts script in the list of Client Scripts and open the script for editing.
2. Select the **Active** check box to make the script active.

3. Examine the pseudo-code for the script:

```
When the value for Impact changes.
    Store the current State in the variable incState.
    Log a message.
    If Impact has the value High or Medium
        Make the Assigned To field mandatory.
        If the user has the itil role
            Generate an alert confirming the itil role.
            Remove the State choice list options Awaiting Problem, Awaiting
            User Info, and Awaiting Evidence.
            Log a message.
            Set the State value to Active.
        Else if the form is not loading and Impact does not have the
        value High or Medium
            If the user has itil role
            Try
                Clear the State choice list.
                Add back options to the State choice list.
                Set the State field to the value of incState.
            Catch JavaScript errors
                Add an Error Message to the top of the form.
```

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.

Testing

1. Launch the JavaScript Log by selecting the bug icon ().

2. Create a new Incident.

3. Select the Clear Log icon () to remove all log messages from JavaScript Log.

4. Watch the State field by right-clicking on the State field on the Incident form and selecting Watch – 'state'.

5. Prepare the Field Watcher.

    a. Select the Field Watcher tab and de-select (uncheck) the All option.

    b. Select the Client script option.

    c. Select the Clear Log icon.

6. Open the State Choice list and notice the options. Do not change the value of the State field.

7. Change the Impact to **1 – High**.

8. Did the Debugging Client Scripts script trigger? How can you tell? Explain your reasoning: (Hint: Examine the State Choice List options as well as the Field Watcher.)

9. From the Field Watcher, can you determine what caused the State field value to change? Explain your reasoning:

10. How can you determine the previous value of the State field? Explain your reasoning:

11. Change the value of the State field to Resolved. Why is there no new entry in the Field Watcher? Explain your reasoning:

12. Switch to the JavaScript Log. Is the State value displayed correctly in the logged message? If not, correct the error in the script. Test the corrected script. Repeat this step until the State value displays correctly in the jslog() statement.

13. Open an existing Incident with an Impact value of anything *except* 3-Low.

14. Clear the JavaScript Log.

15. Change the Inciden'ts Impact field value to **3-Low**.

16. Examine the JavaScript Log for errors. Correct any errors in the script, re-test, and debug until there are no errors.

17. With the JavaScript Log open, impersonate Beth Anglin.

18. When you impersonated Beth Anglin what happened to the JavaScript Log and Field Watcher? Why did this happen?

19. Impersonate the System Administrator.

20. Open any existing Incident.

21. On the Incident form, click the clock icon in the lower right of the frame to open the Response Time Indicator.

22. Examine the response times.

    a. What was the total Response time?

    b. Where was the majority of the Response time spent: network, browser, or server?

23. Open the browser times by selecting the browser Response time link.

    a. How long did it take for the Client Scripts initial load?

    b. How long did it take to load the On Load Client Scripts?

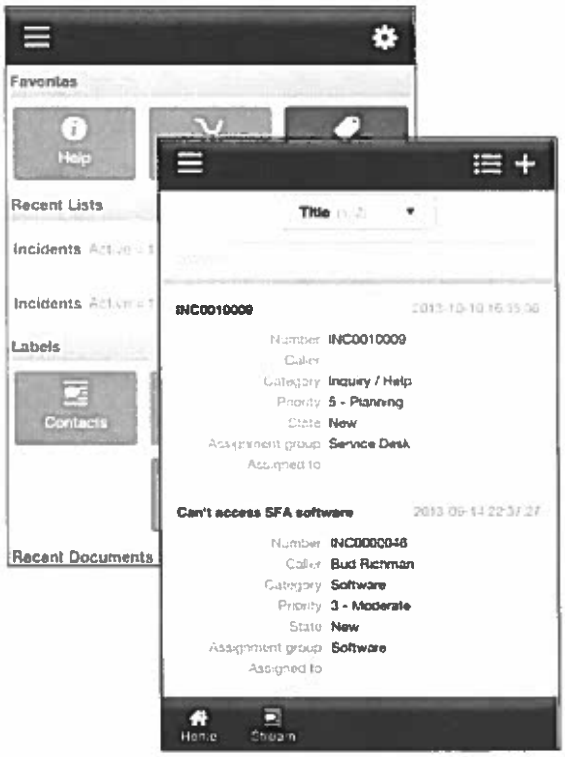
    c. How many Client Scripts loaded? Which scripts loaded?

24. Make the Debugging Client Scripts script inactive.

Mobile UI

servicenow

- ServiceNow runs natively on mobile platforms for supported devices:
  - iPhone
    - iOS6+
    - Safari
  - Android
    - OS 4+ (ICS)
    - Chrome



To run the mobile mode on a desktop browser, append /$m.do to the instance URL.

Mobile UI on Desktop

servicenow

- ServiceNow has a built in Mobile UI emulator
- Append /$m.do to the ServiceNow URL

Although using the Mobile UI emulator gives you an idea of the mobile experience, it is always recommended that you test on mobile devices to experience the software the same way your users will.

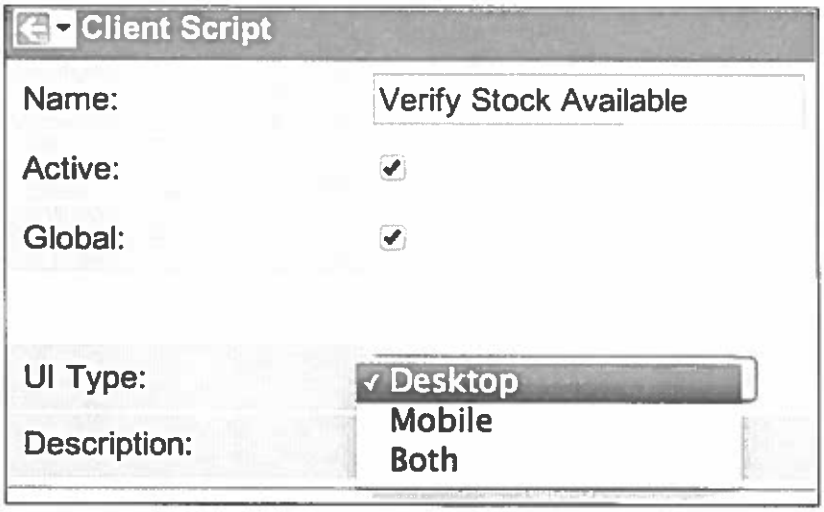
To return to the Desktop mode, open the Application Navigator and select the Logout button at the bottom of the application list. Open <instance URL>/navpage.do.

To add the Mobile UI Emulator to the Application Navigator:
Open System Definition > Appliation Menus.
Open the System Mobile UI Application Menu for editing.
Change the Active value for the Launch the Mobile UI module to true.

Client Script Trigger – UI Type

servicenow

- Specify the platform(s) on which the Client Script executes
  - Desktop = Desktop + Tablet
  - Mobile = iOS and Android smartphones
  - Both = Desktop + Tablet + supported smartphones

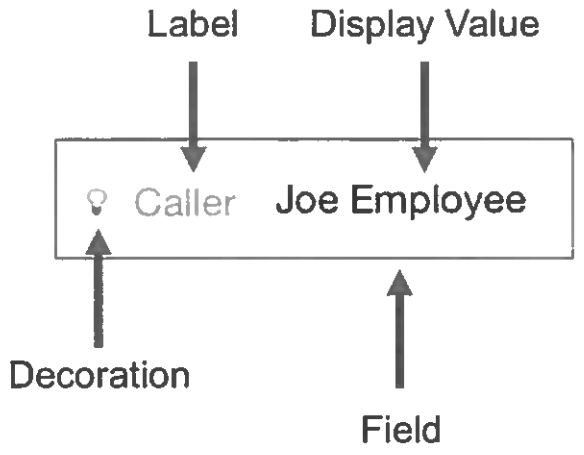


The default Client Script UI Type is Desktop.

Client Script Mobile Methods

servicenow

- g_form methods for mobile
  - setLabel()
  - getLabel()
  - hasField()
  - addDecoration()
  - removeDecoration()
  - getDisplayValue()



g_form methods for mobile:
g_form.setLabel(): sets a field's label text
g_form.getLabel(): gets a field's label text
g_form.hasField(): returns true if a field exists on a form
g_form.addDecoration(): add a decoration next to a field
g_form.removeDecoration(): removes a decoration from a field
g_form.getDisplayValue(): retrieves a field's display value

CAUTION: Using mobile only g_form methods on the Desktop causes runtime errors.

g_form Methods Unavailable for Mobile

servicenow

- Due to limitations of the mobile platform, these g_form methods are not available for Mobile Client Scripts:
  - showRelatedList()
  - hideRelatedList()
  - showRelatedLists()
  - hideRelatedLists()
  - flash()
  - getSections()
  - enableAttachments()
  - disableAttachments()

These methods are not deprecated and are available for use in Desktop Client Scripts. If these methods are used in Mobile Client Scripts, no action is taken.

In addition, these browser objects are not supported in Mobile Client Scripts:

- Window
- jQuery or Prototype ($, $j, $$)
- Document

NOTE: The mobile platform ignores unsupported methods.

Debugging Client Scripts - Mobile

servicenow

- ServiceNow Tools
  - –g_form.addInfoMessage()
  - –g_form.addErrorMessage()
  - –g_form.showFieldMsg()
- JavaScript Tools
  - –Alert
  - –Try/Catch

The JavaScript log and Field Watcher tools are Desktop only tools. To debug Mobile Client Scripts, use the g_form methods and JavaScript alerts.

The g_form methods and alert arguments can be:

- Strings
- g_form properties or methods
- g_user properties or methods
- Variables
- JavaScript string escape characters such as \n (new line) (ignored for g_form methods)

Remember to remove debugging messages when a script is ready for test and production.

Lab

servicenow

3.4 Client Scripting for Mobile

Lab Goal

In this lab you will practice writing, testing, and debugging a Client Script for the Mobile platform.

Lab 3.4
Client Scripting for Mobile

Preparation

Mobile Incident Form

1. Add the Impact and Urgency fields to the Mobile View on the Incident form.

Client Script

1. Open the Client Script Objects Client Script you wrote in the g_form and g_user lab.
2. Make the script active.
3. Set the UI Type to **Both**.
4. Add these JavaScript statements to the script between the alert statement and the g_form.showFieldMsg Statement:

```
g_form.flash('priority','tomato',-8);
```

5. Save.

User – Beth Anglin

1. Open the User record for Beth Anglin and set her password. Record the password value here:

Procedure

1. On your desktop browser, impersonate Beth Anglin.
2. Create a new Incident and set both Urgency and Impact to High.
3. Save the Incident.

4. When prompted whether or not you want to submit the Priority 1 Incident, select Cancel.

5. Did the Priority field flash when you selected Cancel? If not, debug and re-test.

6. Open your instance on a smartphone (if not available, use the Mobile UI emulator) and log in as beth.anglin.

7. Create a new Incident and set both Urgency and Impact to High.

8. Save the Incident.

9. When prompted whether or not you want to submit the Priority 1 Incident, select Cancel.

10. Did the Priority field flash when you selected Cancel? Did you expect it to? Explain your reasoning:

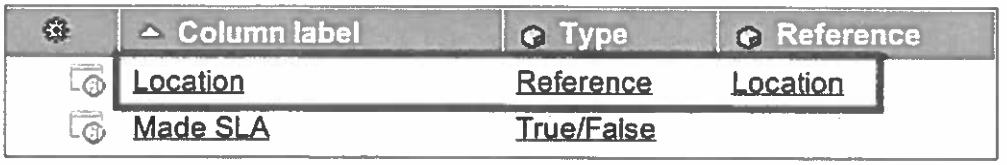
11. Make the Client Script Objects script inactive.

Reference Objects

servicenow

- Reference Object records exist on a table other than a form's currently loaded table

Reference field on a form



Reference field on the table definition

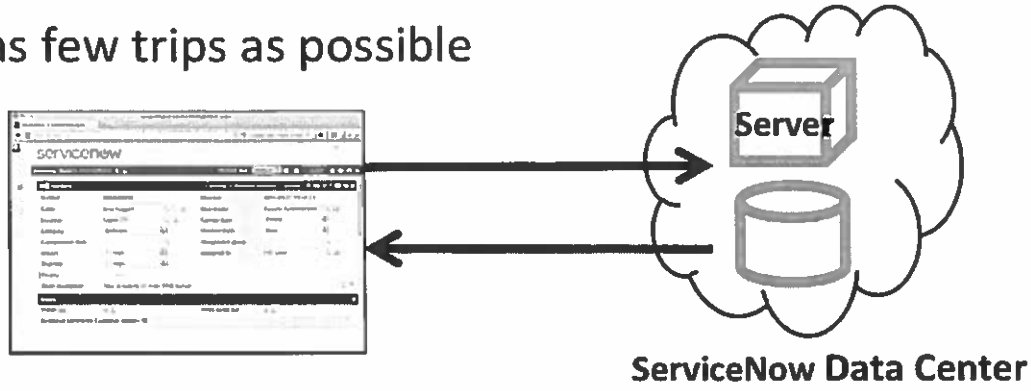
- Reference Object data is not loaded into forms
- Client side scripts only have access to data on forms

When writing client side scripts, you have access to all form fields and their values. Fields of type reference object exist on forms but the reference object record is not loaded into the form.

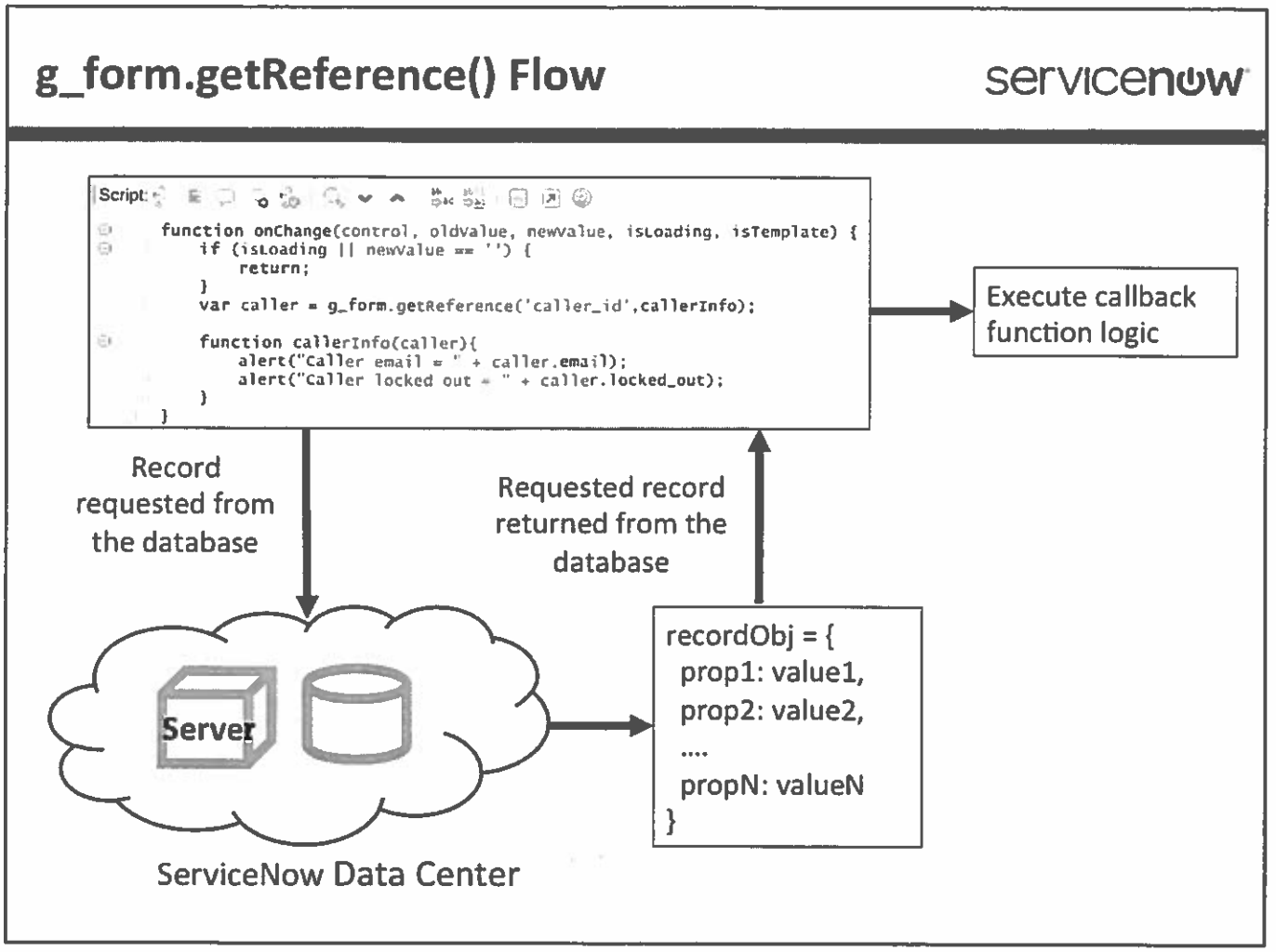
Scripting with Reference Objects

servicenow

- Forms store a Reference Object's sys_id only
- Reference Object fields cannot be directly referenced from a Client Script
- Retrieving Reference Object fields requires a trip to the server and back
  - –Server trips take time
  - –Make as few trips as possible



g_form.getReference() Flow



1. A client side script requests a record from the database.
2. The server retrieves the record from the database and passes the record back to the calling client side script as an object.
3. The callback function logic executes when the object is received from the server.

The g_form.getReference() method runs asynchronously when a callback function is used. When the record is returned to the calling Client Script the logic in the callback function executes. If the callback function is omitted, the g_form.getReference() method runs synchronously.

g_form.getReference Example

servicenow

- The g_form.getReference() method returns a Reference Object's record to a Client Script
- Syntax: g_form.getReference(fieldname,callback)

```
function onChange(control, oldValue, newValue, isLoading,
isTemplate) {
   if (isLoading || newValue == '') {
      return;
   }
  var caller = g_form.getReference('caller_id',callerInfo);

  function callerInfo(caller){
     alert("Caller email = " + caller.email);
     alert("Caller locked out = " + caller.locked_out);
  }
}
```

In the example shown, the callerInfo() function is called only after the caller_id record is returned from the server.

Synchronous vs. Asynchronous

servicenow

- Synchronous (bad practice and won't work mobile)

```
var caller = g_form.getReference('caller_id');
alert("Caller email = " + caller.email);
alert("Caller locked out = " + caller.locked_out);
```

- Also synchronous (bad practice and won't work mobile)

```
var callerEmail = g_form.getReference('caller_id').email;
alert("Caller email = " + callerEmail);
```

- Asynchronous (do this)

```
var caller = g_form.getReference('caller_id',callerInfo);
function callerInfo(caller){
   alert("Caller email = " + caller.email);
   alert("Caller locked out = " + caller.locked_out);
}
```

Synchronous g_form.getReference() method calls lock users out from using a form until the requested record is returned from the database. Asynchronous execution allows users to continue to use forms while the method call executes.

The first example executes synchronously resulting in a poor user experience. g_form.getReference() without a callback function will not run on the Mobile platform.

The second example also executes synchronously and returns only one field's value. The performance penalty is the same as for the first example.

The third example demonstrates how to make an asynchronous call to the server. This strategy works on both the Desktop and Mobile platforms.

Lab

servicenow

3.5 Client Scripting with Reference Objects

Lab Goal

In this lab you will practice using the g_form.getReference() method to retrieve records for reference objects. You will write a script to automatically set the Priority, Risk, and Impact to Low if the Configuration item is 3D Pinball.

Lab 3.5 Client Scripting with Reference Objects

Writing the Script

1. Create a new Client Script.
2. Configure the Client Script Trigger.

   Name: **Reference Objects**
   Active: **Selected** (checked)
   Global: **Selected** (checked)
   UI Type: **Desktop**
   Type: **onChange**
   Table: **Change Request**
   Inherited: **Not selected** (not checked)
   Field: **Configuration Item**

3. Here is the pseudo-code for the script:

   When the Configuration Item field on a Change Request is changed
   Request the CI record from the Server and execute the callback function
   In the callback function
   If the CI is '3D Pinball'
   Set the Priority to Low, Risk to None, and Impact to Low
   Make the Priority, Risk, and Impact fields Read Only

4. Write the script:

```
function onChange(control,oldValue,newValue,isLoading,isTemplate) {
  if (isLoading || newValue == '') {
    return;
  }
  var affectedCI = g_form.getReference('cmdb_ci',checkCI);
```

```
function checkCI(affectedCI){
  if(affectedCI.name == '3D Pinball'){

    g_form.setValue('priority',4);
    g_form.setValue('risk',5);
    g_form.setValue('impact',3);

    g_form.setReadOnly('priority',true);
    g_form.setReadOnly('risk',true);
    g_form.setReadOnly('impact',true);
  }
 }
}
```

5. Submit the script.

Testing

1. Open or create a Change Request:

   Category: **Software**
   Configuration Item: **3D Pinball**

2. Are the Priority, Risk, and Impact fields set correctly? Are they read only? If not, debug the script.

3. Change the Configuration Item to anything except 3D Pinball.

4. Are the Priority, Risk, and Impact fields read only? Should they be? Modify the script using the strategy of your choice so that the Priority, Risk, and Impact fields are read only if the Configuration Item is 3D Pinball and editable for all other Configuration Items.

5. Save the script.

6. Test and debug.

7. Modify the script to display the Requested by user's email address in the Description field by adding a second g_form.getReference() method call and callback function.

8. Save the script.

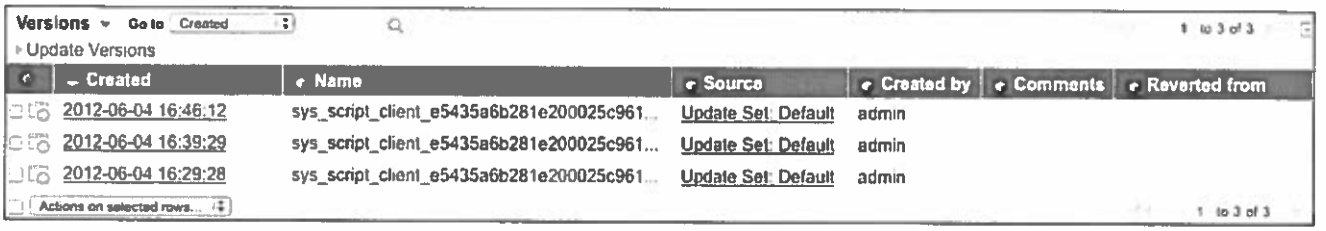
9. Test and Debug.

10. Make the Reference Objects script inactive.

Script Versions

servicenow

- Script versions created automatically on Save, Submit, and Update
- Can compare versions to see differences
- Can revert to a previous script version



Script versions are available for other script types and are not unique to Client Scripts.

Update sets do not include versions. If you migrate a script from one instance to another, only the most recent version is migrated.

Comparing Versions

servicenow

- Select the check boxes for two versions
- Expand the Actions menu at the bottom of the list and select Compare

```
9:    alert('affectedCI = 'affectedCI.manufacturer);
10:
11:   function setValues(affectedCI){
12:     if(affectedCI.manufacturer == 'Cinematronics'){
13:
14:     alert('I am in the if')
```

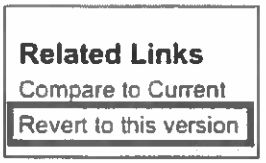
```
9:
10:   function setValues(affectedCI){
11:     if(affectedCI.manufacturer == 'Cinematronics'){
12:       g_form.setValue('priority',4);
```

Red = Deletion
Green = Additions
Yellow = Modification

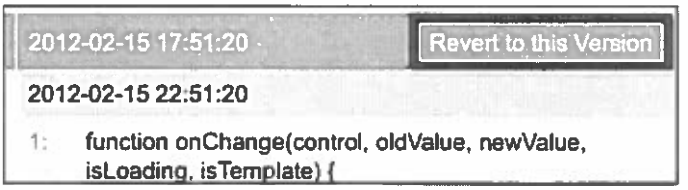
Reverting Versions

servicenow

- Select the version's link to open it for viewing
- Select the Revert to this version Related Link



- When comparing versions select the Revert to this Version button



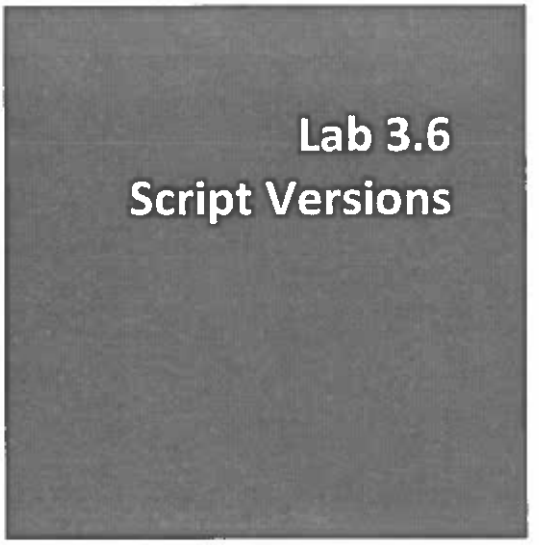
Lab

servicenow

3.6 Script Versions

Lab Goal

In this lab you will examine the differences between scripts and will practice reverting to a previous version of a Client Script.



Reviewing Script Versions

1. Open the Client Script of your choice from the labs you have done so far in class.
2. Scroll to the Versions section. If there are not at least two versions of the script, select a different script for inspection.
3. Select two script versions.
4. Select **Compare** from the Actions on Selected rows menu.
5. Examine the scripts. Has anything been added to or deleted from the current version? Are the versions identical? How can you tell?

6. Revert to the older version of the script by selecting the **Revert to this Version** button on the script comparison window.
7. Look at the Versions list. Can you tell which version has been reverted? Explain your reasoning.

Best Practices

servicenow

- Use the g_form methods to manage the form and form fields
- Use the g_user properties and methods to access information about the currently logged in user
- Make as few calls to the server as possible
- Do not make synchronous calls using g_form.getReference()
- Use jslog to debug as it does not impact other users
- Use try/catch to find runtime errors
- Use methods and debugging strategies appropriate to Mobile, Desktop, or both
- Comment your scripts!

Module Recap

servicenow

Core Concepts

Client Scripts execute in browsers

Use jslog and the JavaScript Log to locate script errors on Desktop

The g_form object has access to a form's fields and data

The g_user object has access to information about the currently logged in user

Reference records are retrieved with the g_form.getReference() method and a callback function

Design your scripts for Mobile, Desktop, or both

Real World Use Cases

Why you would use this

When you would use this

How often you would use this

Discuss: Why, when and how often would you use the capabilities shown in this module?

UI Policies

Module 4

Objectives

Define what it means to be a UI Policy

Know when to use UI Policies

Write, test, and debug UI Policies

Discuss Client Scripts vs. UI Policies

Labs

4.1 Incident State Resolved UI Policy

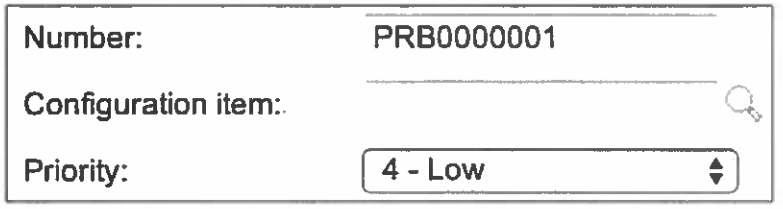
servicenow

In this module you will write, test, and debug UI Policies.

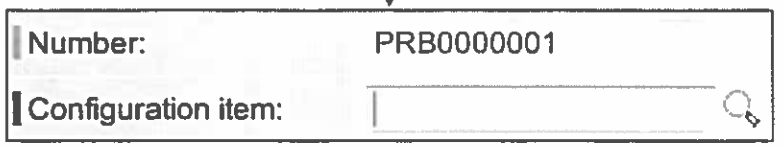
What is a UI Policy?

servicenow

- Client side logic governing form and field behavior
- Have a condition which must be true in order to execute
- Defines the behavior and visibility of fields on a form
  - Mandatory
  - Visible
  - Read Only



Baseline Problem form



Problem form after UI Policy Applied

In the basic case UI Policies do not require scripting. In the example shown, no scripting was required in order to:

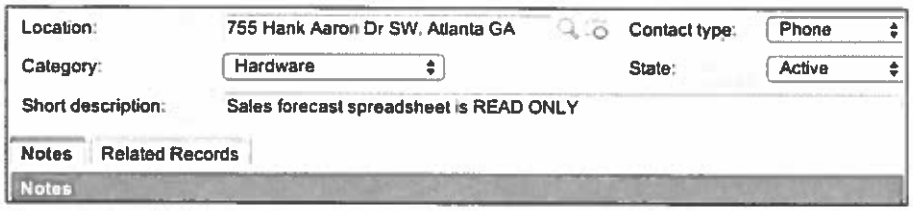
- Make the Number field read only
- Make the Configuration item field mandatory
- Hide the Priority field

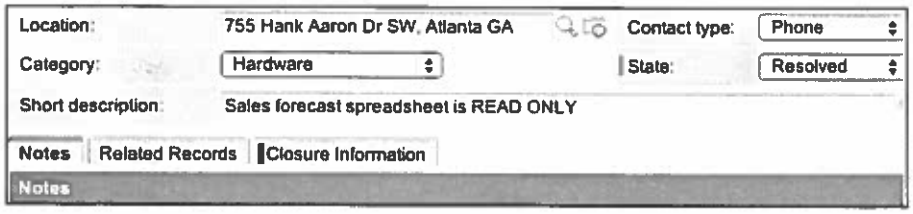
Client Scripts can also hide fields, show fields, and make fields mandatory. Always use a UI Policy instead of a Client Script if you can for faster load times.

What is UI Policy Scripting?

servicenow

- Ability to create complex conditions
- Show/Hide sections (tabs)
- Remove/add/change/validate data in fields
- Full power of JavaScript





Example: UI Policy scripts hide the Closure Information tab unless the State is Closed or Resolved

UI Policy scripts execute on the client side.

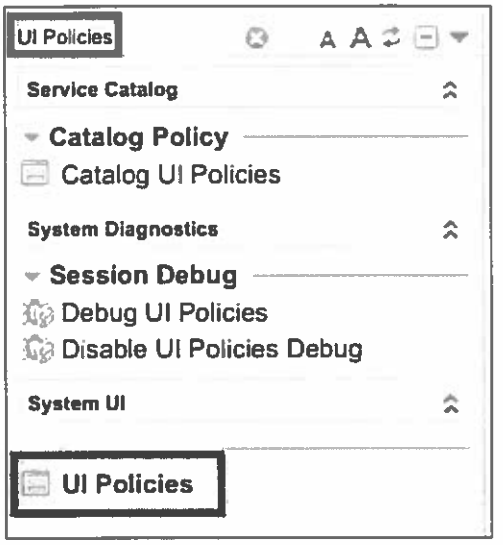
Opening the UI Policy List

servicenow

1. If configured, select UI Policies from the Edge

OR

1. Enter **UI Policies** in the Type Filter text field.
2. Select **System UI > UI Policies**



Another option is to use right click on a form header and select **Personalize > UI Policies** to see the UI Policies for that table.

What Exists Baseline?

servicenow

- Approximately 430 UI Policies exist baseline
- Run whenever Condition met

Short description	Table	Conditions
Type == File	sys_data_source	type=File^EQ
Type == JDBC MID Server	sys_data_source	type=JDBC^EQ
(empty)	sys_dictionary	elementISEMPTY^EQ
(empty)	sys_dictionary	elementISEMPTY^ORinternal_type=boolean^EQ
(empty)	sys_data_source	format=oracle.jdbc.OracleDriver^type!=LD...
(empty)	sys_certificate	type=key_store^ORtype=pkcs12_key_store^EQ
(empty)	sys_script	action_run_at=client^EQ
(empty)	sys_dictionary	internal_type=reference^EQ
Hide fields for display rules	sys_script	when=before_display^EQ
Show priority for async rules	sys_script	when=async^EQ
Hide View when Global checked	sys_ui_policy	global=true^EQ

UI Policies do not have a Name field. When debugging, use the Short description field to determine which UI Policy executed.

The Active column is not in the UI Policies List layout baseline.

UI Policy Trigger

servicenow

- UI Policies execute in known order based on value in Order field; lower the number, earlier it executes
- Assign different order numbers to ensure execution order
- Always include a short description

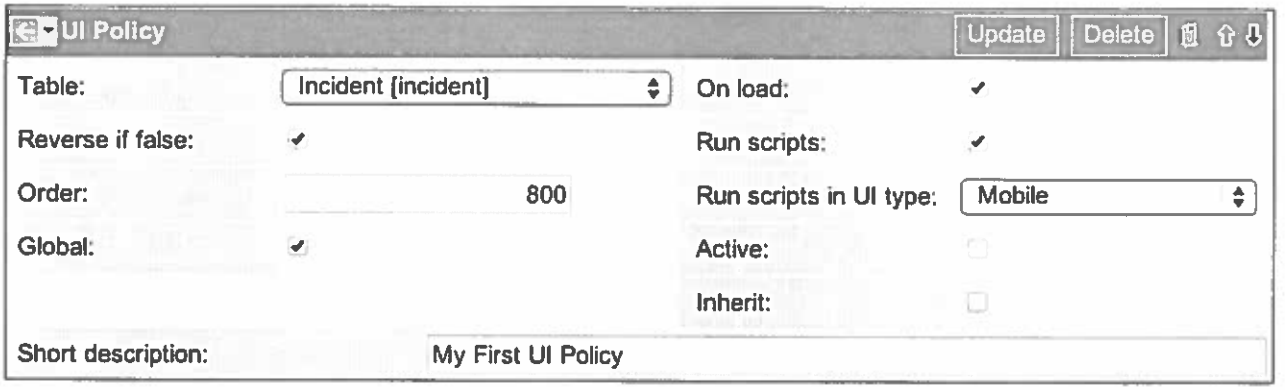


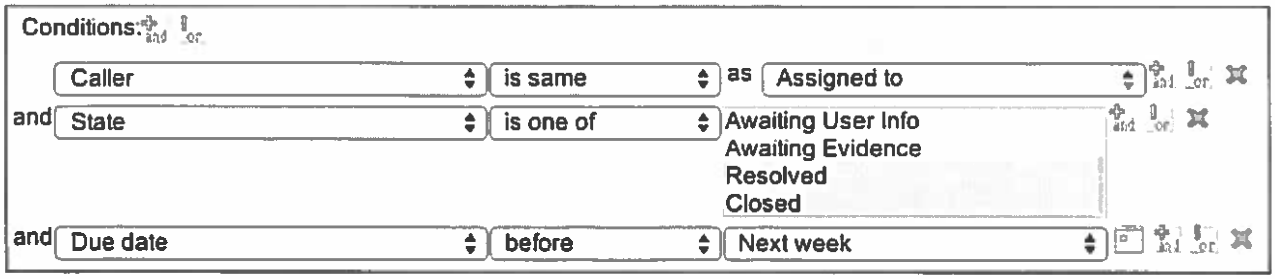
Table: Table to which the script applies.
Reverse if false: UI policy actions are reversed and Execute if false script executes when its UI policy's conditions evaluate to false.
Order: Sequence in which UI policies are applied, from lowest to highest.
Global: The UI Policy applies to all views if selected.
View: The name of the view to which the script applies.
On load: Execute on form load *and* form change.
Run scripts: Allows scripts to run for true and false conditions.
Run scripts in UI type: Select whether the script executes for Desktop and Tablet or Mobile or both. Only visible when Run scripts is selected.
Active: Select the check box to make the UI Policy active.
Inherit: Apply this script to any extended tables when selected.

If the Global option is not selected you must specify the view to which the script applies. The View option is only visible when Global is not selected. Select the view before writing a script. A script can only act on fields that are part of the selected form view. If you do not select Global and leave the View field blank the script applies to the default view.

UI Policy Trigger Conditions

servicenow

- UI Policies execute based on evaluation of their Condition
  - Build Conditions with the Condition Builder rather than scripting for better performance
  - If blank the UI Policy logic will always execute



The Condition Builder is not unique to UI Policies: filters, survey administration, SLAs, List Filters, Report Conditions etc.

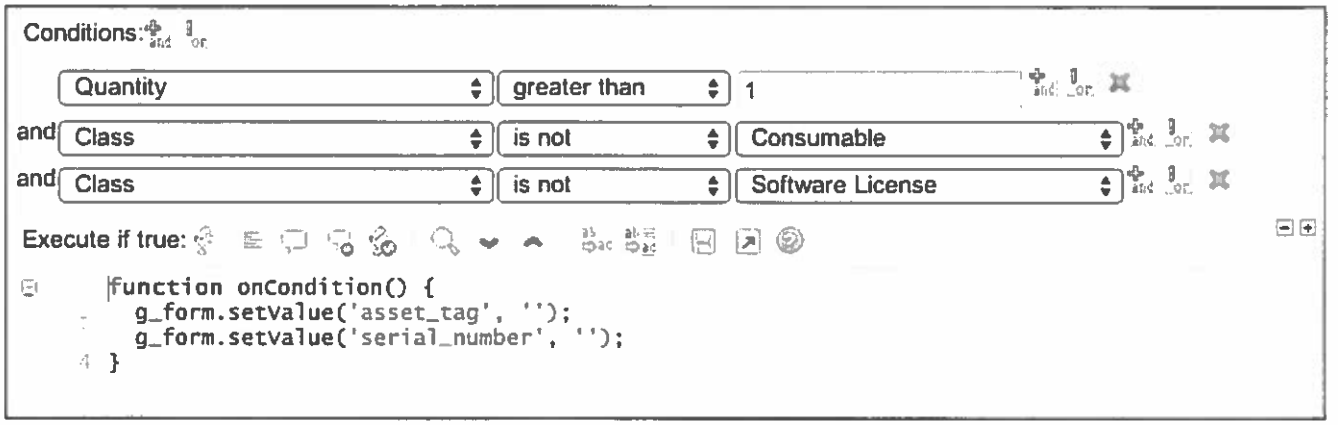
Conditions are only rechecked if a user manually changes a field for a record on a form; if the change is made by a UI Action, Context Menu action, from an import set, via web services, or through the List Editor it will not be evaluated. Use Data Policies, which are not scriptable, to manage the mandatory and read only state of fields for records not changed on a form.

The field must be on the form to be checked by a UI policy. In order to test the value of a field but hide the field from users, add the field to the form, and use a UI policy to hide it from the users' view.

Execute if True

servicenow

- JavaScript that runs when the UI Policy condition tests true

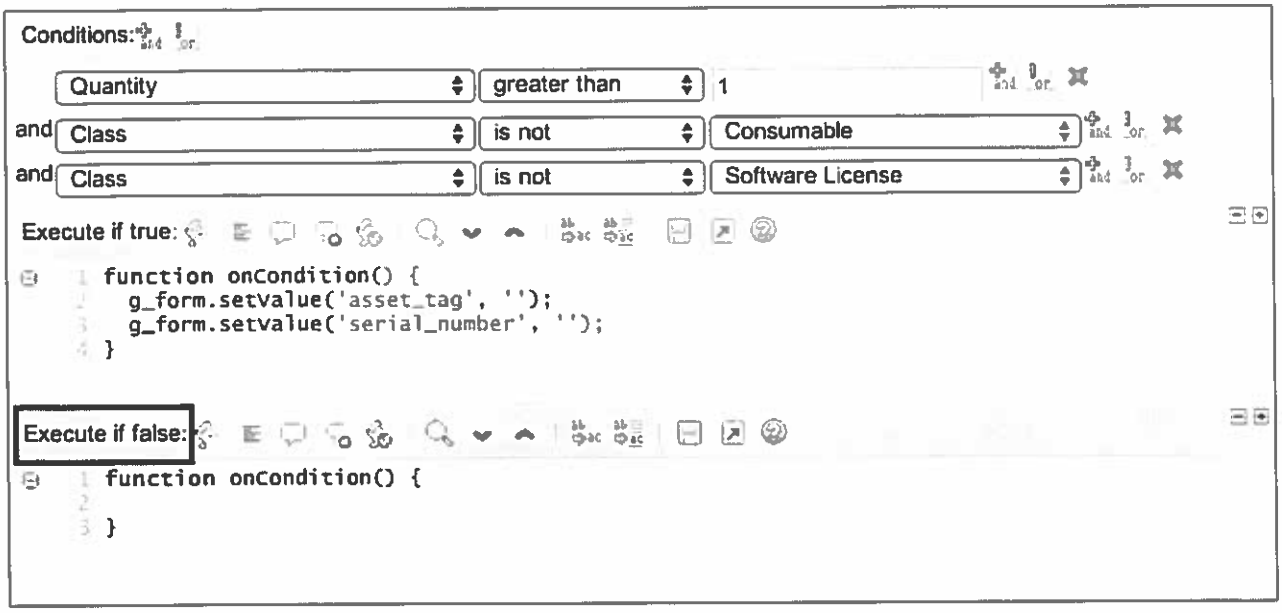


The onCondition() function in the UI Policy script fields is automatically called when the condition returns true or false.

Execute if False

servicenow

- JavaScript that runs when the UI Policy condition tests false



In the example shown, no action is taken by the UI Policy when the condition tests false.

A UI Policy with Run scripts selected may contain an Execute if true script, an Execute if false script, or both. Do not select the Run scripts option if no scripts are included in the UI Policy definition as this attaches unnecessary administrative overhead to the UI Policy execution.

What data can I use in a UI Policy?

servicenow

- Local variables declared in script
- Client Script Global Variables
  - –g_form
  - –g_user
  - –g_scratchpad

ServiceNow client side global variables:

- g_form is an object whose properties are the fields from the currently loaded form. The property values are the field values from the form.
- g_user is an object whose properties contain session information about the currently logged in user and their role(s).
- g_scratchpad is a global object passed to a client script from a server side script. The object's properties and values are typically determined by the server side script.

Debugging UI Policies

servicenow

Desktop

- ServiceNow Tools
  - Debug UI Policies
  - JavaScript Log
  - Field Watcher
  - Response Time Indicator
- JavaScript Tools
  - Try/Catch

Mobile

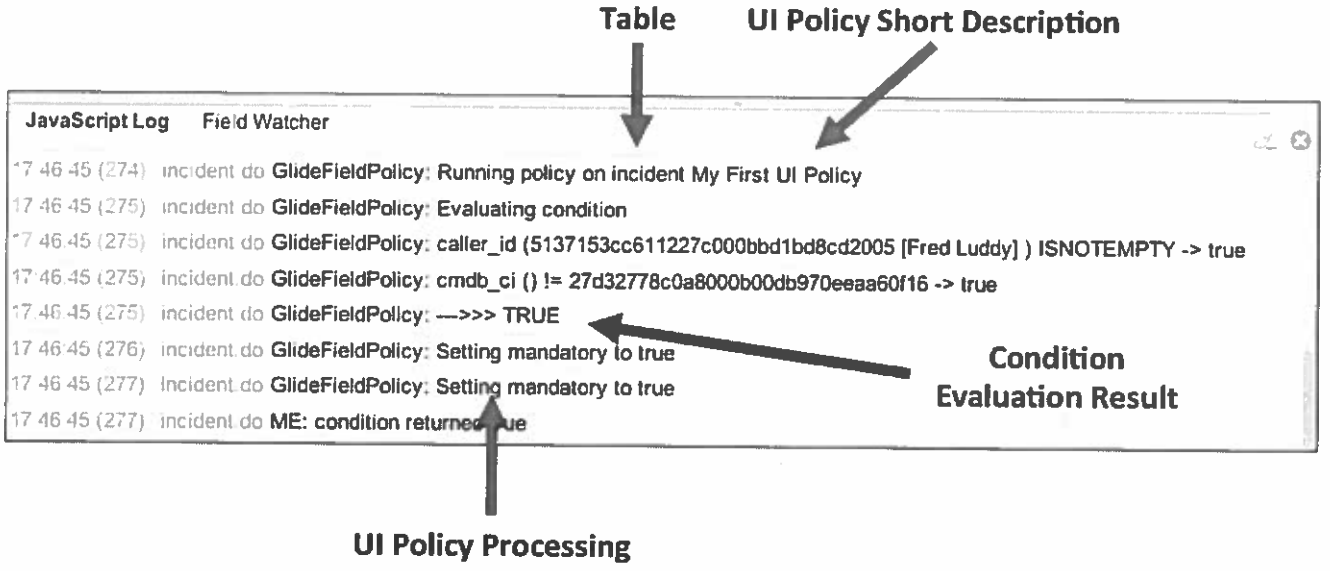
- ServiceNow Tools
  - Info/Error messages
- JavaScript Tools
  - Alerts
  - Try/Catch

UI Policies execute their scripts on the client side and all of the debugging strategies that work for Client Scripts also work for UI Policies. In addition, UI Policies have the Debug UI Policies option.

Debug UI Policies

servicenow

- Must be enabled/disabled
- Output appears in the Javascript Debug Window
- Shows evaluation of condition and script processing



Select **System Diagnostics > Session Debug > Debug UI Policies** to enable UI Policy debugging.

Select **System Diagnostics > Session Debug > Disable UI Policies Debug** to disable UI Policy debugging.

Debug UI Policies is the only debugging strategy for seeing the evaluation of a UI Policy's Condition.

Lab

servicenow

4.1 Incident State Resolved UI Policy

Lab Goal

In this lab you will write, test, and debug a UI Policy. When an incident's State is set to Resolved:

- Configuration Item, Urgency, and Impact are read only.
- Closed by is mandatory
- Generate an alert to notify the user that the incident is resolved.

Lab 4.1
Incident State Resolved UI Policy

Configure UI Policy

1. Enter **UI Policy** in the Type Filter text field.
2. Click, hold, and drag the **System UI > UI Policies** module to the Edge.
3. Open the UI Policies list view.
4. Create a new UI Policy.
5. Configure the UI Policy trigger:

   Table: **Incident [incident]**
   Reverse if false: **Selected** (checked)
   Order: **150**
   Global: **Selected** (checked)
   Short Description: **Incident's State is Resolved**
   On load: **Not selected** (not checked)
   Run Scripts: **Not selected** (not checked)
   Active: **Selected** (checked)
   Inherit: **Not selected** (not checked)

6. Set the Condition:

   State is Resolved

7. Save (not Submit) the UI Policy.

8. Add a new UI Policy Action to the UI Policy by selecting the **New** button at the bottom of the form.

9. Configure the UI Policy Action:

   Table: **incident**
   Field name: **Configuration item**
   Mandatory: **Leave alone**
   Visible: **Leave alone**
   Read Only: **True**

10. Submit the UI Policy Action.

11. Add three additional UI Policy Actions:

    Urgency: **Read only**
    Impact: **Read only**
    Closed by: **Mandatory**

12. Modify the trigger:

    Run Scripts: **Selected** (checked)
    Run Scripts in UI type: **Both**

13. Save (not Submit) the UI Policy.

14. Examine the pseudo-code for the Execute if True script you will write:

    When the condition is true Generate an alert stating that the Incident's State is resolved. Request the user complete the Close Notes and Closed By fields.

15. Write the Execute if True script:

```
function onCondition() {
    alert("You changed the incident's State to Resolved. \n\nPlease
    complete the Close Notes and the Closed By fields.");
    }
```

16. Update the UI Policy.

17. Test the UI Policy:

    a. Open an Incident
    b. Set the State to **Resolved**.

18. Did the alert appear?

19. Are the correct fields hidden or mandatory?

20. Enable UI Policy Debugging: **System Diagnostics > Session Debug > Debug UI Policies.** Open the Javascript Debug window and force the UI Policy to execute again. Examine the debug output. Did your UI Policy execute as expected? Troubleshoot and debug any problems with the UI Policy.

21. Does this UI Policy need an Execute if False script? Why or why not?

22. Test the UI Policy on the mobile platform.

    a. Did the alert appear?

    b. Are the correct fields hidden or mandatory?

23. Disable the UI Policy by de-selecting the Active check box in the Trigger.

24. Disable UI Policy Debugging: **System Diagnostics > Session Debug > Disable UI Policies Debug**

Client Scripts vs. UI Policies

servicenow

	Client Script	UI Policy
Execute on form load	✔	✔
Execute on form save/submit/update	✔	
Execute on form field value change	✔	✔
Have access to a field's prior value	✔	
Execute on list field value change(s)	✔	
Control the order of execution		✔
Execute after Client Scripts		✔
Require scripting	✔	

Client side scripts manage forms and their fields. Client Scripts and UI Policies both execute client side and use the same API. Use the table to determine which script type is best suited to your application needs.

Best Practices

servicenow

- Set onLoad to false if you don't need to execute on page load
- Use as few UI Policies as possible to avoid long page load times
- Write conditions in the Condition Builder whenever possible so unnecessary scripts don't load
- Always populate the Short Description field to document the UI Policy

Consider adding the Description field to the UI Policy form to thoroughly document the UI Policy.

Module Recap

servicenow

Core Concepts

UI Policies are used for client side form management

UI Policies take actions based on scripts and/or UI Policy Actions

Debug UI Policies using logging, JavaScript, and UI Policy Debugging

Different actions are available if the condition returns true or false

Always populate the short description field

Real World Use Cases

Why you would use this

When you would use this

How often you would use this

Discuss: Why, when and how often would you use the capabilities shown in this module?