Scripting in servicenow

In ServiceNow, **scripting** refers to writing and using code to customize and automate processes, tasks, and business logic within the platform. It enhances the functionality and behavior of ServiceNow applications beyond the out-of-the-box capabilities.

ServiceNow primarily uses two types of scripts:

1. **Client-side scripting**: Runs on the user's browser. It is used to control how forms, fields, and user interfaces behave. Common client-side scripts include:
2. **Server-side scripting**: Runs on the server and is used to control data operations and business logic. Common server-side scripts include:

These scripts use **JavaScript** as the programming language and allow developers to manipulate data, automate tasks, validate data, create integrations, and more.

Client-side vs. Server-side

Scripts in ServiceNow execute either on the client (user's browser) or in the ServiceNow back end. It is important to know where a script will execute as there are different APIs for the client and server-side scripts.

Client-side

- Manage forms and form fields:
- UI Policies
- Client Scripts

Server-side

- Manage database and back-end:
- Business Rules
- Scheduled Jobs
- Script Actions
- Script include

The script types listed here are examples; this is not an exhaustive list of script types in ServiceNow.

Client Scripts

It is a client side script, it is used to make configurations and changes to the forms

ServiceNow client scripts execute in a browser. Client scripts manage forms and form fields in real time. You can:

- Modify choice list options.
- Set one field in response to another in real time.
- Hide/Show form sections.
- Display an alert.
- Make fields mandatory.
- Hide fields.

onLoad() client scripts are typically used to pre-populate fields with values and make other form appearance and content modifications. While onLoad() client scripts executes, users have no ability to modify form fields.

This script runs when a form meeting the trigger condition loads and before control is given to the user.

onSubmit() client scripts are typically used the perform data validation against multiple fields simultaneously. Users have no access to a form's fields while onSubmit() client scripts execute. This script runs when a form meeting the trigger condition is saved, updated, or submitted.

onChange() client scripts are used to validate a field's value in real time or to set or modify field values in response to another field changing.

Onload()

Executed on loading the form

1. Pre populate
2. Content modifications
3. Form appearance

Use onLoad client scripts sparingly as they impact form load times.

Onchange() onChange client scripts execute script logic when a particular field's value changes.

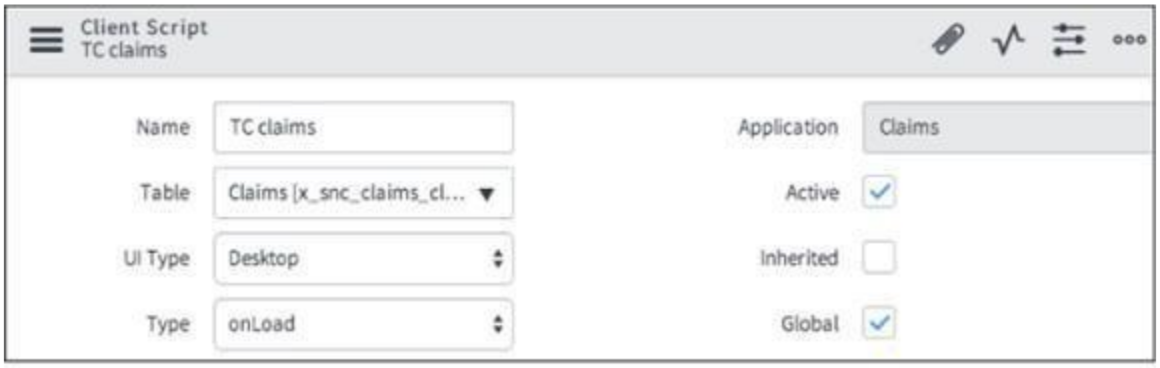
Use onChange client scripts to respond to field values of interest and to modify another field's value or attributes. For example, if the State field's value changes to Closed Complete, generate an alert and make the Description field mandatory.

Onsubmit() onSubmit client scripts execute script logic when a form is submitted.

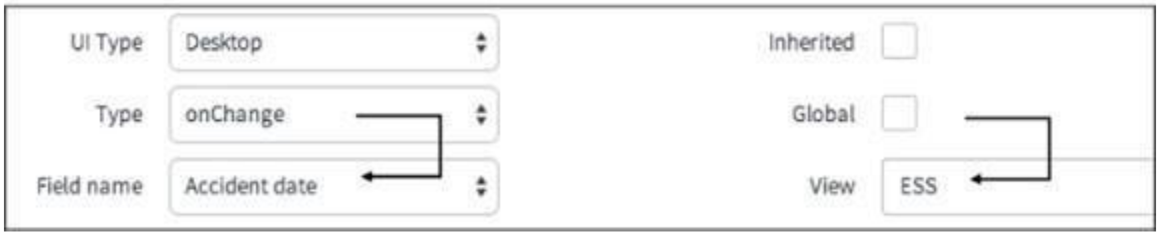
Use onSubmit client scripts to validate field values. For example, if a user submits a Priority 1 record, the script can generate a confirmation dialog notifying the user that the executive staff are copied on all Priority 1 requests.

onLoad client scripts will only run if the page is reloaded because the user saved the page. If a user submits or updates the page, after the data is sent to the server, users are redirected back to their previous page.

The trigger condition must be met for the client script to execute.


Client Script
TC claims

Name	TC claims	Application	Claims
Table	Claims [x_snc_claims_cl...	Active	✓
UI Type	Desktop	Inherited	
Type	onLoad	Global	✓



UI Type	Desktop	Inherited	
Type	onChange	Global	
Field name	Accident date	View	ESS

Name: Name of client script. Use a standard naming scheme to identify custom scripts.

Table: Form to which the script applies.

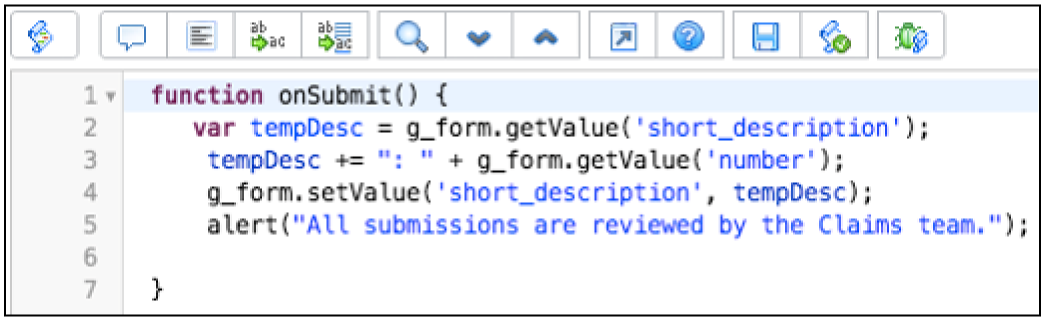
UI Type: Select whether the script executes for Desktop and Tablet or Mobile or both.

Type: Select when the script runs: onChange, onLoad, onSubmit, or onCellEdit.
GlideForm

The g_form object is automatically instantiated for client-side scripts. There are g_form methods that are specific to desktop/tablet and methods that are specific to mobile.

Properties are the fields from a form’s table.

- Property values are field values.
- Contains methods for managing form fields and values.



```
function onSubmit() {
   var tempDesc = g_form.getValue('short_description');
    tempDesc += ": " + g_form.getValue('number');
    g_form.setValue('short_description', tempDesc);
    alert("All submissions are reviewed by the Claims team.");

}
```

Give users feedback through Client Scripts and UI Action scripts.

- g_form.showFieldMsg() - prints a message on a blue background below the field passed in the method call.
- g_form.addInfoMessage() - prints a message on a blue background to the top of the current form.
- g_form.addErrorMessage() - prints a message on a red background to the top of the

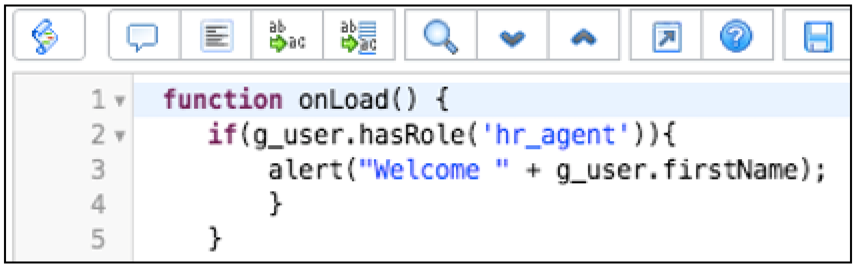
current form.

- alert is a JavaScript method that opens a dialog with an OK button.
- confirm is a JavaScript method that opens a dialog with OK and Cancel buttons.
- g_form.setMandatory()
- g_form.SetReadOnly()
- g_form.setdisplay()
- g_form.set_value(' feild_name" , 'value') - sets the value of the field
- g_form.get_value('feild_name) - returns the value in the field

GlideUser

The g_user object is automatically instantiated for client-side scripts.

- Property/Values are information about the current user.
- Contains methods for checking roles for the current user. ● Used to personalize the user experience.



```
function onLoad() {
    if(g_user.hasRole('hr_agent')){
        alert("Welcome " + g_user.firstName);
        }
    }
```

- G_user.firstname
- G_user.lastname
- g_user.userId
- G_user.username
- G_user.getfullname
- g_user.hasrole('role_name')
- g_user.hasroleexactly('role_name')
- g_user.hasroleFromList('role_name')
- g_user.hasroles()

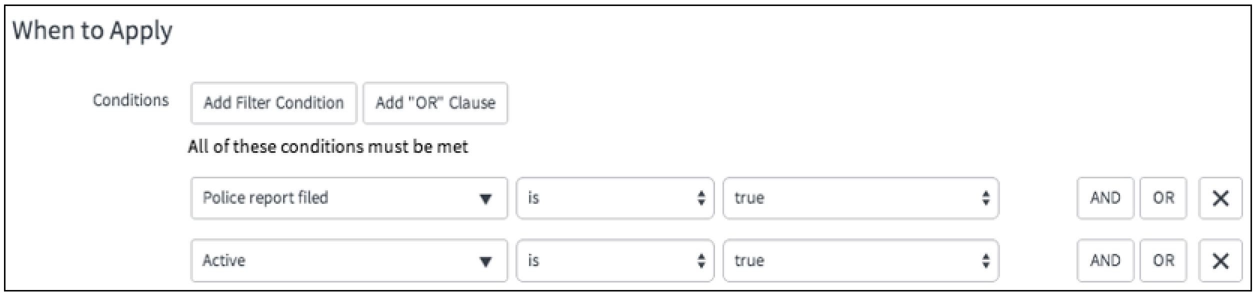
Because g_user executes client-side, impersonation will impact results. For example, if Beth Anglin has the hr_agent role, and the System Administrator impersonates Beth Anglin to test the onLoad script shown, the alert will say "Welcome Beth".

UI policy

Client-side logic governing form and field behavior. UI Policies dynamically change the behavior of information on a form and control custom process flows for tasks. you can use UI policies to make the number field on a form

- read-only,
- field mandatory
- hide other fields.

UI Policies execute based on evaluation of their Condition. For better performance, use the Condition Builder to build conditions rather than scripting. If blank, the UI Policy logic will always execute.


When to Apply

Conditions: Add Filter Condition | Add "OR" Clause

All of these conditions must be met

Police report filed	is	true
Active	is	true

Ui policy does not have a name, they have on description From the context menu configure the UI Policy Actions:

Field name – field for which the UI Policy applies an action.

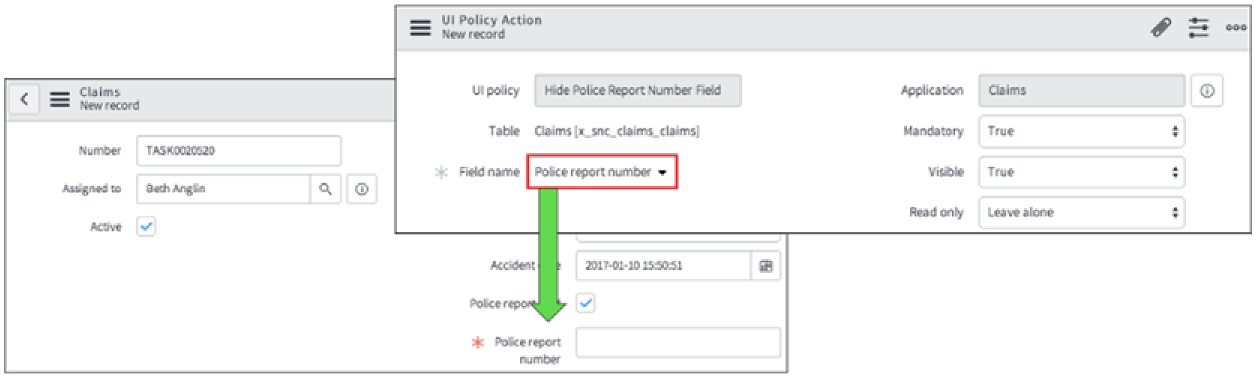
Mandatory – how the UI policy affects the mandatory state of the field.

Visible – how the UI policy affects the visible state of the field.

Read only – how the UI policy affects the read only state of the field.

steps to create a UI Policy Action:

1. In Studio, open **Client Development > UI Policies** and open a UI Policy for editing.
2. Scroll to the UI Policy Action section.
3. Select the New button.
4. Configure the UI Policy Action.
5. Save.


UI Policy Action
New record

UI policy	Hide Police Report Number Field	Application	Claims
Table	Claims [x_snc_claims_claims]	Mandatory	True
Field name	Police report number	Visible	True
		Read only	Leave alone

Claims
New record

Number	TASK0020520
Assigned to	Beth Anglin
Active	✓
Accident	2017-01-10 15:50:51
Police report	✓
Police report number	

UI Policy scripts execute on client-side, with full use of client-side JavaScript methods.

The script tab becomes available in the Advanced view of the UI Policy form. Initially, the only item on the tab is the Run Scripts checkbox. Once checked, other fields appear. Developers can write a script in one or both script editors.

- Run scripts: Select the Run scripts option to access the Scripting fields.
- Execute if true: JavaScript that executes when the UI Policy Condition tests true.
- Execute if false: JavaScript that executes when the UI Policy Condition tests false.

Server-Side Scripting in ServiceNow

In ServiceNow, server-side scripts are written on the server. The scripts process data and business logic, and then what they have processed is relayed to the user's browser. The core idea of applying business rules, data manipulation, and other system interactions all revolve around server-side scripts.

Server-Side Classes

GlideRecord : Contains methods for querying and manipulating records within a database

GlideSystem : The utility class for multiple functions, such as logging and retrieving system properties.

Comparison Client-Side Classes:

GlideForm: Manages form data on the client side, used in UI scripts.

GlideUser: Returns information about the currently logged-in user.

GlideAjax: Lets any client-side script communicate with server-side scripts.

Data Policy

Data consistency is enforced across ServiceNow applications by a Data Policy because it specifies whether fields are mandatory or read-only. Unlike UI Policies, which may only be able to influence data submitted through forms, Data Policies influence everything-from submitting through import sets to through web services.

Important Details:

Mandatory Fields: Those fields marked as mandatory fields need to be filled in before a record can be saved.

Read-Only Fields: Fields marked as read-only can only be viewed and not changed.

Server and Client-Side Execution: Data Policies run on the server side; however, it may have client-side implications as well when data is updated through forms.

Objective:

All input source must have standardized ways of creating data.

The field requirements are constant, so it maintains data integrity

Business Rule

A ServiceNow Business Rule is a server-side script that is executed upon any database operation: insertion, update, delete, or query. Using business rules allows for the automatic enforcement of business logic server-side.

Setup:

Name: Provide a name to the business rule.

Application: Limit the scope of the business rule to avoid unintentional impact

Table: Choose the relevant data table on which the business rule will be applied

Active : Enable or disable the business rule.

Advanced: Click the advanced button for script options.

Order: Specify the order the Business Rules will execute from low to high.

Types of Execution:

Insert: This one executes when a new record is added to the database

Update: A present record in the database is being updated

Delete: Deletes a record

Query: Runs where a record is queried, exclusively with a database view

When to Run:

Before: This script runs before your data is actually written into the database, thereby allowing input data being modified.

After: Runs after saving the data, is useful for procedures that rely on the last state of the record.

Display: Runs whenever preparing data to be displayed within a form, is useful when attaching additional information to a form.

Async: Runs in the background after the record was saved, improves the performance by offloading responsibilities of the main transaction.

Execution Flow:

Trigger Condition: It can be triggered by business rules over any action performed on a table (like insert, update).

Conditions: State under what condition the Business Rule should be executed.

Script Execution: If the conditions are fulfilled, then the script gets executed and modifications/extension also on server-side operations can be done.

Best Practice:

Documenting Business Rules: Include description of Business Rules for better comprehension and maintainability

Server-side scripting in ServiceNow involves Data Policies and Business Rules. Each has a distinct role in ensuring consistency in data, enforcing business logic, and automating processes. Its understanding along with its configurations is vital for effective development and administration within ServiceNow.

Script Include in ServiceNow

A Script Include is a reusable server-side JavaScript class or function developed to centralize code and make it available across multiple parts of the ServiceNow platform. This helps increase code modularity, reusability, and maintainability by storing functions and logic that can be invoked from other server-side scripts, like Business Rules and Workflow scripts, as well as from client-side scripts via GlideAjax.

Key Features of Script Includes:

Reusable JavaScript: Script Includes is the central location for defining reusable JavaScript functions.

Modularity: Encourages code reuse, thus making it less redundant

Debugging: Kept related functions in one place, thus reducing debugging complexity

Initialization: Allows an initialization function to prepare the required data before calling a function.

Anatomy of a Script Include

Name:

This is the unique identifier for the Script Include. Only alphanumeric characters will do. No spaces or special characters allowed. Use a consistent naming convention so it's easier to read.

API Name:

Full Name of the Script Include which indicates its scope and the script name.

It is used when calling the Script Include from other scripts.

Client Callable:

Check box indicating if the Script Include can be called from client-side scripts by using GlideAjax

Only permit this if it must be enabled for security reasons.

Application:

Determines the scope of the Script Include

This subsequently identifies what applications can access it.

Accessible From:

Applied to All Scopes: Incorporates global allowing access from any script within the system .

This Application Scope Only: Restricts access only to include scripts within the same application scope as a form of encapsulation

Active:

Active checkbox that needs to be checked before the Script Include becomes executable.

In case it is left unmarked, then the Script Include will never execute even when other scripts make calls to it

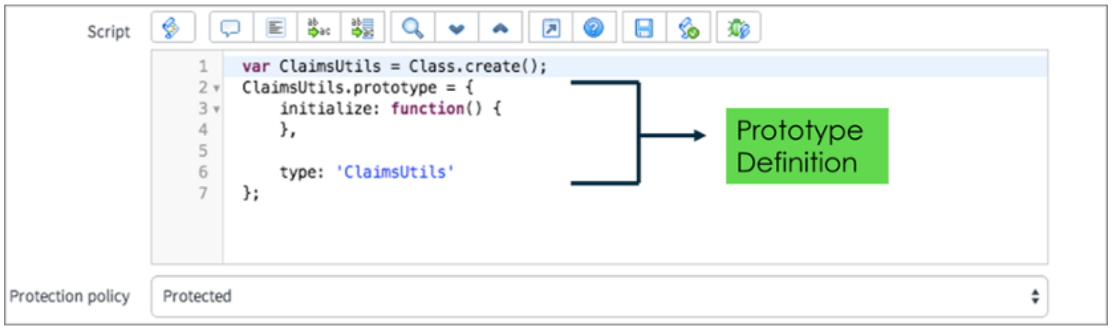
Description:

This is an optional but highly recommended field to comment the purpose and function of the Script Include.

Keeps any future maintainers and developers in context.

Script Include Structure

A script usually consists of a class definition which contains methods and properties which described its functionality.

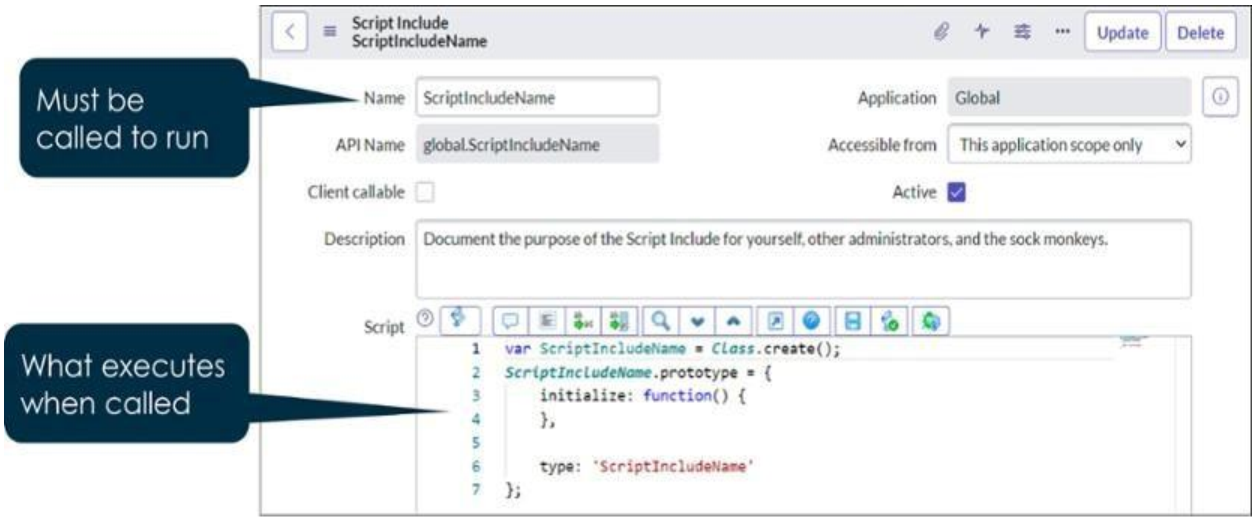


```
var ClaimsUtils = Class.create();
ClaimsUtils.prototype = {
    initialize: function() {
    },

    type: 'ClaimsUtils'
};
```

Protection policy: Protected

},



Script Include
ScriptIncludeName

Name	ScriptIncludeName	Application	Global
API Name	global.ScriptIncludeName	Accessible from	This application scope only
Client callable	☐	Active	✓
Description	Document the purpose of the Script Include for yourself, other administrators, and the sock monkeys.		

Script

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

    type: 'ScriptIncludeName'
};
```

Types of Script Include

Script Includes can be categorized into two types based on how they are structured:

- Classless Script Include (Function-Based Script Include)
- Class-Based Script Include (Object-Oriented Script Include):
- Extend an existing class

Classless Script Include (Function-Based Script Include):

These Script Includes contain simple, standalone functions rather than being encapsulated in a class.

var myFunction = function(param) {

return 'Hello, ' + param;

};

var result = myFunction('John'); gs.info(result); // Outputs: Hello, John

Class-Based Script Include (Object-Oriented Script Include):

These Script Includes are structured using JavaScript classes (via ServiceNow’s Class.create() method).

var MyClass = Class.create();

MyClass.prototype = {

initialize: function() {

},

```
myMethod: function(param) {

return 'Hello, ' + param;

},

anotherMethod: function() {

return 'This is another method';

},

type: 'MyClass'

};

};
```

Call a Script Include:

1. From a Server-Side Script:

A Script Include is typically called from Business Rules, Script Actions, or any other server-side components.

```
var utils = new IncidentUtils(); // Instantiating the Script Include

var description = utils.getIncidentByNumber('INC0012345'); // Calling the method
gs.info(description); // Log the result
```

2. From a Client-Side Script using GlideAjax:

To call a Script Include from the client-side (like in a Client Script or UI Action), you use GlideAjax.

- First, mark the Script Include as Client Callable by checking the Accessible from Client checkbox.

```
var ga = new GlideAjax('IncidentUtils'); // Script Include name ga.addParam('sysparm_name',
'getIncidentByNumber'); // Method name ga.addParam('sysparm_incidentNumber',
'INC0012345'); // Parameter ga.getXMLAnswer(function(response) {

var answer =

response.responseXML.documentElement.getAttribute("answer"); alert(answer); // Displays
the result in the client

});
```

UI Actions

UI Actions in ServiceNow add interactivity and customization to the user interface by adding buttons, links, and context menu items to forms and lists, which allow the user to perform specific operations right in the UI.

Types of UI Actions

Form Buttons:

Add buttons at the top or bottom of a form.

Example: A "Submit" button on an incident form.

Add entries to the Header Menu when right-clicking a form header.

Example: Custom actions like "Reassign Incident".

Form Links:

Add links in the Related Links section of the form.

Example: link to "View Change Requests", which is relevant to the current incident.

List Buttons:

Adds buttons at the top of a list view.

Example : "New Incident" button on the All Incidents list.

List Context Menu Items:

Adds menu items to the context menu when right-clicking a record in a list.

Example: "Assign to Me" in the context menu of a list item.

List Options:

Provide options at the bottom of a list view for batch actions.

Example: An "Export" option to export list data.

List Links:

Add links to the Related Links section at the bottom of a list view.

Example: A link to "View Related Changes" for selected incidents.

Configuration and Behavior:

Order:

Specifies the position of the UI Action relative to other UI Actions. For example, a UI Action with an Order of 100 will appear before UI Actions that have an order number greater than 100 and after those that have a lower order number.

Active: If this field is selected, the UI Action will be shown and activated for the user to interact with.

Setting Client-Server:

Client: UI Actions with the Client checkbox checked will run client-side scripts and can be invoked from the client browser.

Server: UI Actions with the Client checkbox unchecked will run server-side scripts and will execute at the server

Navigation:

You can access UI Actions by going to All > System Properties > UI Properties.

Sample UI Action Script:

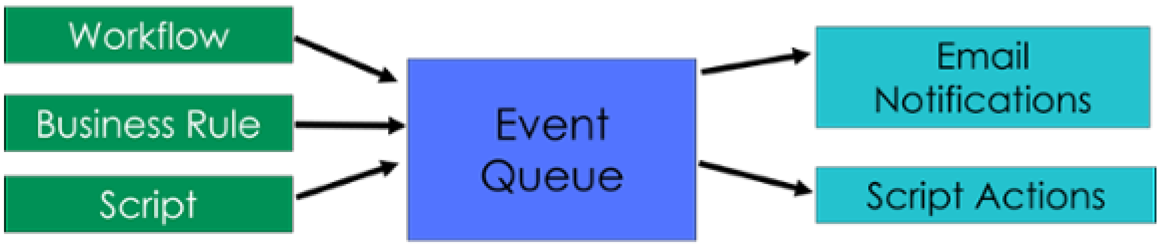
javascript

Copy code

```
// Client-side UI Action script example
function myClientSideFunction() {
   alert('This is a client-side UI Action');
}


// Server-side UI Action script example
(function executeRule(current, previous /*null when async*/) {
   gs.addInfoMessage('This is a server-side UI Action');
})(current, previous);
```

Script Actions



Script Actions are server-side scripts executed in response to specific system events. They form part of the ServiceNow event-driven architecture and are designed to process backend logic in response to triggers stemming from events.

Key Features:

Event-Driven:

Triggered by specific events in the ServiceNow platform.

Server-Side:

Runs on the server instead of on the client-side.

Events of Interest:

Events

It is associated with events in the event queue.

Script Actions Use Cases

Notifications:

#Trigges the notification of events such as when an incident gets resolved.

Update Records:

#Modify or update records based on some specific events.

Logging:

System information or errors about events can be recorded.

Interface with External Systems:

Data can be sent to or received from external systems based on any event.

Methods in Script Actions:ings.eventQueue()

Enqueues a custom event for processing.

```
gs.eventQueue('custom_event', current, 'param1', 'param2');
```

gs.addErrorMessage()

Adds an error message to be displayed to the user.

```
gs.addErrorMessage('An error occurred while processing the request.');
```

current

Refers to the record that is currently being processed.

event

Identifies which event has invoked the Script Action.

Example Script Action Script:

javascript

Copy code

```javascript
// Script Action to send a notification when an incident is closed
(function executeRule(current, previous /*null when async*/) {
if (current.state == 6) { // Assuming 6 is the 'Closed' state
        gs.eventQueue('incident_closed', current, current.number, 'Closed');
    }
})(current, previous);
```

Scheduled Jobs

Scheduled Jobs allow for automating periodic activities through running a script on a given schedule. They are time-driven and not event-driven; thus, they are great for scheduled jobs or batch processing.

Key Features:

Purpose:

Most commonly used for maintenance jobs, data update, batch process, or clean-up jobs.

Schedules:

Can be scheduled to be run on daily, weekly, monthly, and any other custom schedule.

Execution Time:

It runs according to a predefined schedule rather than for real-time events.

Example Scheduled Job Script:

javascript

Copy code

```
// Scheduled Job script to clean up old records
(function executeJob() {
   var gr = new GlideRecord('incident');
   gr.addQuery('sys_created_on', '<=', gs.daysAgo(30));
   gr.deleteMultiple();
})();
```

Each of these allows increases in functionality along with automation of your ServiceNow applications and can be dramatically enhanced with the usage of UI Actions, Script Actions, as well as Scheduled Jobs.

Glide Record

GlideRecord is a powerful API in ServiceNow used to interact with tables in the platform. It allows developers to perform CRUD (Create, Read, Update, Delete) operations and query records from the database. GlideRecord abstracts the underlying database and provides a simple way to manage records.

Key GlideRecord Functions

1. **initialize()**: Initializes the GlideRecord object for a specific table.
2. **addQuery()**: Adds a condition to the query. Multiple queries can be added to filter records based on specific criteria.
3. **query()**: Executes the query defined by the addQuery() conditions and retrieves the matching records from the database.
4. **next()**: Moves the GlideRecord object to the next record in the result set. It is used in loops to iterate over multiple records.
5. **get()**: Retrieves a single record based on a unique identifier (sys_id) or other key value.
6. **insert()**: Inserts a new record into the database for the table specified by the

GlideRecord object.

7. **update()**: Updates the current record in the database with any changes made to the

GlideRecord object.

8. **deleteRecord()**: Deletes the current record from the table.
9. **setValue()**: Sets the value of a specific field in the current GlideRecord object.
10. **getValue()**: Retrieves the value of a specific field from the current record in the

GlideRecord object.

11. **hasNext()**: Checks if there are more records to process in the result set.
12. **setLimit()**: Limits the number of records returned by the query.
13. **orderBy()**: Sorts the result set in ascending order based on a specified field.
14. **orderByDesc()**: Sorts the result set in descending order based on a specified field.
15. **getTableName()**: Returns the name of the table associated with the GlideRecord object.
16. **updateMultiple()**: Updates multiple records that meet the conditions set by the query.
17. **isNewRecord()**: Checks if the current record is new and has not yet been saved to the database.
18. **deleteMultiple()**: Deletes all records that match the query conditions.
19. **getEncodedQuery()**: Retrieves the encoded query string representing the conditions that were added using addQuery().
20. **addEncodedQuery()**: Adds an encoded query string to the GlideRecord query, often used for complex queries.

Uses:

- **Querying Data**: Retrieve records from a table by specifying conditions.
- **Updating Records**: Modify fields of a record and update them in the database.
- **Deleting Records**: Remove records based on specific conditions.
- **Inserting Records**: Add new records to the database.

Glide System

GlideSystem (often abbreviated as gs) is a server-side API in ServiceNow that provides methods and utilities for various operations like logging, user information retrieval, events, and more. It's frequently used in **Business Rules**, **Script Includes**, **UI Actions**, **Scheduled Jobs**, and other server-side scripts to interact with the platform's core functionality.

Key Features of GlideSystem:

1. Logging and Debugging:
   - gs.log(): Logs messages to the system log.
   - gs.error(), gs.warn(), gs.info(): Logs error, warning, and informational messages respectively.
   - gs.debug(): Logs debug information when debugging is enabled.
2. User Information:
   - gs.getUser(): Returns a GlideUser object for the currently logged-in user.
   - gs.getUserID(): Returns the sys_id of the current user.
   - gs.getUserName(): Returns the user name of the current user.
   - gs.hasRole(role): Checks if the current user has a specific role.
   - gs.getUserDisplayName(): Returns the display name of the current user.
3. Date and Time Utilities:
   - gs.now(): Returns the current date and time in the system's timezone.
   - gs.daysAgo(days): Returns the date n days ago from the current date.
   - gs.dateDiff(): Calculates the difference between two dates.
   - gs.addDays(): Adds a specified number of days to the current date.
4. Events:
   - gs.eventQueue(): Triggers an event and pushes it into the event queue for processing.
   - gs.eventQueueScheduled(): Queues an event for future processing at a scheduled time.
5. Security & Permissions:
   - gs.hasRole(): Checks if the current user has a specific role.
   - gs.hasRightsTo(): Checks if the current user has access rights to a specific record.
6. Error and Exception Handling:

- gs.addErrorMessage(): Adds an error message to the session, visible to the user.

- gs.addInfoMessage(): Adds an informational message to the session, visible to the user.

7. Database & Records:
   - gs.getProperty(): Retrieves system properties.
   - gs.setProperty(): Sets system properties.
   - gs.generateGUID(): Generates a unique identifier (GUID).
   - gs.nil(): Checks if a variable is null or empty.

ServiceNow API documentation

locate the API documentation on the ServiceNow Developer website:

1. Log into the developer.servicenow.com website.
2. Open **Reference > API > Client or API > Server.**
3. Select a class from the navigation pane on the left.
4. Select a method.

Passing data from server side to client side we have there methods

Since Client- side scripts such as client scripts and ui policies cannot get access to the server data we use the following to get access to the data from server side to client side

1. G_scratchpad
2. g_form.getreference('feild_name')
3. GlideAjax

Using g_scratchpad

Display Business Rules pass data from the server-side to a client-side script using the g_scratchpad object. All property values must be passed as strings.

Display Business Rule

g_scratchpad.prop1 = current.field1; g_scratchpad.prop2 = current.field2; g_scratchpad.prop3 = current.field3; Client Script g_form.setValue('description',g_scratchpad.prop1);

The g_scratchpad object has no properties by default and must be populated by a Display Business Rule. Any client-side script can use the g_scratchpad object passed in from the server.

Using GlideAjax

GlideAjax is a client-server communication method in ServiceNow that allows client-side scripts (like those in UI Pages, UI Actions, and Client Scripts) to call server-side code asynchronously. It provides a way to fetch data or perform actions on the server from the client without refreshing the entire page.

How GlideAjax Works:

- Client-side script makes an asynchronous call to the server using the GlideAjax object.
- Server-side script (Script Include) performs the necessary operations and returns data back to the client.
- The client then processes the returned data (e.g., updates the UI).

GlideAjax Components

1. **Client-side Script**: The client-side script (like a UI action, client script, etc.) that initiates the GlideAjax call.
2. **Script Include (Server-side)**: The server-side Script Include that processes the request and returns the data.

Example

How to Use GlideAjax

1. Client-side Script

This script will call the Script Include using GlideAjax.

```
var ga = new GlideAjax('UserLookup'); ga.addParam('sysparm_name', 'getUserName');
ga.addParam('sysparm_userID', g_user.userID);

ga.getXMLAnswer(function(response) { var userName =
response.responseXML.documentElement.getAttribute('answer'); alert('The user name is: ' +
userName);

});
```

2. Script Include (Server-side)

The Script Include performs the server-side processing and returns the result. It must be marked as Client Callable for it to work with GlideAjax.

```
var UserLookup = Class.create();
```

```
UserLookup.prototype = Object.extendsObject(AbstractAjaxProcessor, {

getUserName: function() { var userID = this.getParameter('sysparm_userID'); var gr = new
GlideRecord('sys_user'); if (gr.get(userID)) { return gr.name.toString();

}

return 'User not found';

}

});
```

Common Use Cases for GlideAjax:

1. Form Validation:
2. Populating Fields:
3. Dynamic Data Retrieval
4. Interacting with Complex Business Logic