DIGITAL NURTURE 3.0 SERVICENOW LEARNINGS

Week 4 ServiceNow Scripting Fundamentals and Functions

TOPICS COVERED

1. **What is scripting?**
   scripting is an essential part of customizing and extending platform functionality. You can write scripts on both the client and server side.

there are 2 types of scripting one is server side and other is client side

client side scripting isexecuted in the browser, affecting how users interact with forms, fields, and UI elements. One of the main tools for client-side scripting is Client Scripts.

Types of Client Scripts

1. **OnLoad**

Executes when a form is loaded.

```
function onLoad() {

g_form.setVisible('field_name', false);

 }
```

2. **OnSubmit**
   Runs when a form is submitted (when the user clicks on the submit, update, or save button). It's mainly used for validation or to prevent submission based on certain conditions.
   ```
   function onSubmit() {
    var fieldValue = g_form.getValue('field_name');
   if (fieldValue === '')
   { alert('This field is mandatory!');
    return false; // Prevents form submission
    }
   return true; // Allows form submission
    }
   ```
3. OnChange
   Executes when a specific field value is changed.
   ```
   function onChange(control, oldValue, newValue, isLoading) {
   if (newValue !== oldValue && !isLoading)
    { g_form.setValue('other_field', 'Updated Value');
   }
   ```

4.OnCellEdit

Runs when a user edits a field in a list (specifically in a cell in list views).

```
function onCellEdit(sysIDs, table, oldValue, newValue, callback)
 {
 if (newValue !== oldValue) {
('Cell edit detected'); } }
```

Client-Side API provides methods to interact with form fields, user data, and UI components. These APIs make it possible to manipulate form elements, validate input, and handle user interactions.

Key Concepts

- **g_form**: An object that gives access to form fields, allowing you to manipulate field values, visibility, read-only status, etc.
- **g_user**: Allows you to access the user's session information, such as roles and user ID.

Commonly Used g_form Methods:

- **getValue(fieldName)**: Retrieves the value of a specified field.
  setValue(fieldName, value): Sets the value of a specified field.
  isMandatory(fieldName): Checks if a field is mandatory.
  setMandatory(fieldName, mandatory): Makes a field mandatory or non-mandatory.

  setVisible(fieldName, visible): Shows or hides a field.
  showErrorBox(fieldName, message): Displays an error message below a field.

Commonly Used g_user Methods:

- **g_user.hasRole(roleName)**: Checks if the current user has a specific role
  g_user.getFullName(): Retrieves the full name of the current user
  g_user.getUserName(): Gets the username of the logged-in user.
  g_user.getUserID(): Retrieves the user ID of the logged-in user.

The g_scratchpad object allows you to pass data from server-side scripts to client-side scripts. It is used when you need server-side processing but want to utilize the result on the client side.

GlideAjax is used for asynchronous communication between the client and the server. It allows you to call server-side scripts from client-side scripts without refreshing the page.

- **g_form**: Interaction with form fields (e.g., get/set values, show/hide fields).
- **g_user**: Access logged-in user details (e.g., username, roles).
- **g_scratchpad**: Pass data from server to client.
- **GlideAjax**: Communicate with server asynchronously.

There are different mechanisms to enforce rules and policies, both on the client side (UI) and the server side (business rules).

A **UI Policy** is a client-side rule that dynamically changes the behavior of form fields based on certain conditions. It allows you to control the visibility, read-only state, and mandatory state of form fields in real-time as users interact with the form.

Example:

In a Leave Request Form, if the Leave Type is Medical, the Medical Certificate field becomes mandatory, and the Reason for Leave field becomes read-only.

Example:

UI Policy Behavior:

- **Condition**: Leave Type = "Medical"
- **Action**:
  - Set **Medical Certificate** field to mandatory.
  - Set **Reason for Leave** field to read-only.

A **Data Policy** is a server-side rule that enforces data consistency and quality by making fields mandatory or read-only on all data entry points (forms, imports, APIs, etc.).

Example:

You want to ensure that the **Email** field on a **User** table is always mandatory, regardless of whether the data is entered through a form, import, or an API. You can define a data policy to enforce this.

Server-Side Scripting in ServiceNow

Server-side scripting in ServiceNow is used to automate processes, enforce business rules, and ensure data consistency on the server.

Key Objects in Server-Side Scripting:

- **GlideRecord**: Used to query, update, and create records in the database.
- **gs (GlideSystem)**: Provides utility functions like logging, getting the current user, or generating unique IDs.

A **Business Rule** is a server-side script that runs when a record is inserted, updated, deleted, or queried from the database. It is used to enforce rules or execute server-side logic based on the data being processed.

Business Rule: Server-side logic that runs on insert, update, delete, or query operations in the database. Can be "Before", "After", "Async", or "Display".

1. GlideRecord API

GlideRecord is the primary API used to interact with database tables in ServiceNow. It allows you to perform CRUD (Create, Read, Update, Delete) operations on records.

Key Methods in GlideRecord:

- **Initialize and Insert**: Create new records in a table.
- **Query and Get Records**: Retrieve data from tables.
- **Update and Delete**: Modify or remove records.
- **Query Conditions**: Add filters to narrow down the search.
- **Query Conditions**

  When querying the database, you often need to filter records based on certain conditions. **GlideRecord** provides methods like `addQuery()`, `addEncodedQuery()`, and others to build complex queries.
  addQuery(field, operator, value)
  This method adds a condition to the query.

An **Encoded Query** is a condensed string representation of query conditions. Instead of adding individual conditions with addQuery(), you can use an encoded query string for more complex queries.

Example:

Retrieve incidents where the priority is **1 - Critical** and the state is either **New** or **In Progress** using an encoded query.

```
var gr = new GlideRecord('incident');
gr.addEncodedQuery('priority=1^stateIN1,2'); // Encoded Query: Priority = 1 AND State IN (1, 2)
gr.query();
while (gr.next()) {
  gs.log(gr.number);
}
```

- ^: Represents AND between conditions.
- **IN**: Represents multiple possible values (comma-separated).

GlideDateTime is a class used to handle date and time in ServiceNow. It provides methods to manipulate, compare, and format date/time values.

Common Methods in GlideDateTime:

- **getDisplayValue()**: Returns the date and time as a string in the user's time zone.
- **getValue()**: Returns the date and time as a string in UTC format.
- **addDays(days)**: Adds a specified number of days to the date.
- **subtract(startDateTime)**: Subtracts a date from the current date to get the difference.

1. Getting Current Date and Time

```
var gdt = new GlideDateTime(); gs.log(gdt.getDisplayValue());
```

2. **Add Days to a Date**

```
var gdt = new GlideDateTime(); gdt.addDays(5); // Adds 5 days to the current date gs.log(gdt.getDisplayValue());
```

Scenario: Auto-Assigning Incidents to Support Groups Based on Category

In many organizations, incidents in ServiceNow are categorized (e.g., Hardware, Software, Network). Each category is handled by a specific support group. We want to automate the assignment of incidents to the appropriate support group when a user selects a category.

We'll implement this solution using a Before Business Rule that runs when an incident is created or updated. It will check the Category field and assign the appropriate Support Group.

Step 1: Create a Before Business Rule

1. Navigate to System Definition > Business Rules.
2. Create a new business rule with the following parameters:
   - Table: Incident
   - When to run: Before (Before insert or update)
   - Condition: If the Category field changes.

Condition:

```
current.category.changes();
```

Step 2: Business Rule Script

Write the script to check the incident category and set the **Assignment Group**.

```
(function executeRule(current, previous /*null when async*/) {
    // Create a mapping of categories to assignment groups
    var categoryToGroup = {
        'Hardware': 'Hardware Support Group',  // sys_id or name of the
group
        'Software': 'Software Support Group',
        'Network': 'Network Support Group'
    };

    // Get the category of the incident
    var category = current.category;

    // Check if there is a corresponding support group for the category
    if (categoryToGroup[category]) {
        // Assign to the corresponding support group
        current.assignment_group = categoryToGroup[category];
} else {
        // If no match, assign to default IT Support group
        current.assignment_group = 'IT Support Group';  // sys_id or name
    }
})(current, previous);
```

- A user submits an incident with the category Hardware.
- The Before Business Rule runs before the record is saved and assigns the incident to the Hardware Support Group.
- If the user submits an incident with an uncategorized option, it defaults to the IT Support Group.