2110030140cse@gmail.com

T.Leekhitha

**Digital Nurture-3.0 Week-4**

**1)Understand the fundamentals of scripting on the ServiceNow platform.**

**Ans**: To understand the fundamentals of scripting on the ServiceNow platform, it's essential to learn about the different types of scripts used, the contexts in which they are applied, and how ServiceNow handles them. Here's a breakdown of the key concepts:

**1. Client-Side Scripting**

Client-side scripts execute in the browser of the user interacting with the ServiceNow instance. They are generally used for user interface (UI) and cosmetic changes.

- **Client Scripts**: These scripts run in response to actions on the form, such as loading, changing fields, or submitting data.

- Types of Client Scripts:

- onLoad(): Runs when the form loads.

- onChange(): Runs when a form field changes.

- onSubmit(): Runs before the form is submitted.

- onCellEdit(): Runs when a cell in a list changes.

- **UI Policies**: Control the behavior of fields (such as visibility or mandatory conditions) based on user interactions. Though UI policies are primarily configuration-based, advanced use cases can involve scripting.

- **UI Actions**: Buttons, links, and context menu items that trigger scripts when a user interacts with the UI.

**Example**: Client Script (onLoad)

```javascript

function onLoad() {

g\_form.setValue('short\_description', 'This is a sample default value');

}

**2. Server-Side Scripting**

Server-side scripts run on the ServiceNow server and handle data processing, querying databases, and backend logic.

**- Business Rules**: Automatically execute in response to database operations (create, update, delete, etc.). They run when a record is inserted, updated, or queried.

- **Types**:

- Before: Executes before an insert or update operation.

- After: Executes after the database operation is completed.

- Async: Runs after the record has been saved to the database and is processed in the background.

- **Script Includes**: Reusable server-side scripts that can be called from other server-side scripts or workflows. These are modular and efficient.

- **GlideRecord:** This is a special API used to query and manipulate records in ServiceNow.

**Example**: Business Rule (After Insert)

javascript

(function executeRule(current, previous) {

var gr = new GlideRecord('incident');

gr.get(current.sys\_id);

gr.short\_description = 'Record created: ' + current.number;

gr.update();

})(current, previous);

**3. Glide APIs**

ServiceNow provides Glide APIs that make it easier to interact with the platform's tables and data. The most commonly used Glide objects include:

- **GlideRecord**: For querying, updating, and creating records.

- **GlideAggregate:** For performing database aggregation operations (like counting, summing).

- **GlideDateTime:** For handling date and time values in scripts.

Example: GlideRecord Query

```javascript

var incident = new GlideRecord('incident');

incident.addQuery('priority', 1);

incident.query();

while (incident.next()) {

gs.info('High-priority incident: ' + incident.number);

}

**4. Scripting in Workflows**

In addition to direct client-side and server-side scripting, workflows can also involve scripting in ServiceNow. For example:

- **Flow Designer**: A low-code tool used to automate processes. Though scripting isn't required, action steps can involve scripts.

- **Workflow Script**s: Scripts in workflows that run during specific stages of the workflow process.

**5. Other Advanced Scripting Topics**

- ACL (Access Control List) Scripting: Controls what users can see and do based on roles and conditions. Often requires server-side scripting.

- Email Scripts: Scripting to control the content and behavior of outgoing email notifications.

- Integration Scripts: Scripting for REST or SOAP integrations, using APIs to connect ServiceNow with external systems.

**6. Service Portal and Widget Scripting**

ServiceNow’s Service Portal provides an interface for users to interact with the platform. Widgets in the portal allow for custom user experiences, combining:

- HTML (for the structure)

- Client-side JavaScript (for UI behavior)

- Server-side JavaScript (for data processing)

- Client-side scripting focuses on UI behavior and operates in the user's browser.

- Server-side scripting handles backend logic such as database queries, data processing, and business rules.

- Use Glide APIs for interacting with ServiceNow's database and records.

- Understand how scripting fits into the overall structure of ServiceNow, from UI policies to integrations, and from business rules to email notifications.

**2)Learn how to script effectively to customize and enhance ServiceNow functionalities.**

**Ans** To script effectively in ServiceNow, it's crucial to understand the platform's architecture, the key areas where scripting is applied, and how to make the most of ServiceNow’s APIs. Below are the essential steps and practices to improve your scripting skills and leverage the platform's capabilities to enhance its functionalities:

**1. Understand the Key Scripting Areas in ServiceNow**

ServiceNow uses scripting in various contexts. Knowing these will help you understand where to customize and extend functionalities:

- Client-side scripting: Involves form validation, UI actions, field manipulation, and dynamic behavior.

- Server-side scripting: Handles business logic, database operations, data integrity checks, and record processing.

- Scripting in Workflows and Flow Designer: For process automation and orchestration.

- Scripting in Service Portal: For customizing and building user-friendly interfaces.

**2. Master the Glide API**

The Glide API is the most important scripting interface in ServiceNow, allowing you to interact with the database, records, and other platform features. Mastering this API is crucial to effective scripting.

- GlideRecord: Used to query and manipulate database records.

- GlideAjax: For calling server-side code from client scripts.

- GlideDateTime: For date manipulation and time zone handling.

- GlideUser: Provides user-related information.

- GlideSystem (gs): Common utility methods for logging, event handling, and notifications.

**Example**: GlideRecord Query and Update

```javascript

var incident = new GlideRecord('incident');

incident.addQuery('priority', 1);

incident.query();

while (incident.next()) {

gs.info('High-priority incident found: ' + incident.number);

incident.state = 2; // Change state to "In Progress"

incident.update();

}

**3. Efficient Use of Client-Side Scripting**

Client-side scripting improves the user experience (UX) by dynamically interacting with forms. These scripts are usually lightweight but should be optimized for performance.

- Client Scripts: Use them for validation, field visibility, and form manipulation. Minimize complexity in client-side scripts to avoid performance bottlenecks.

- UI Policies: Whenever possible, use UI Policies instead of Client Scripts for field-level control to reduce the need for custom scripting.

**Example**: Client Script for Field Auto-fill

```javascript

function onLoad() {

var user = g\_user.getFullName();

g\_form.setValue('description', 'Assigned to: ' + user);

}

**4. Understand and Use Asynchronous Processing**

Asynchronous processing is important to prevent delays or performance issues, especially with server-side scripts.

- Async Business Rules: Use async rules to offload tasks that do not need to happen immediately (e.g., sending notifications after a record is created).

- GlideAjax: Use it for client-to-server communication without blocking the client.

Example: GlideAjax for Client-Server Communication

Client Script:

```javascript

function fetchServerData() {

var ga = new GlideAjax('CustomScriptInclude');

ga.addParam('sysparm\_name', 'getData');

ga.getXMLAnswer(function(response) {

var answer = response.responseXML.documentElement.getAttribute('answer');

g\_form.setValue('description', answer);

});

}

```

Script Include:

```javascript

var CustomScriptInclude = Class.create();

CustomScriptInclude.prototype = Object.extendsObject(AbstractAjaxProcessor, {

getData: function() {

return 'Server-side data fetched!';

}

});

**5. Scripting Best Practices**

To enhance ServiceNow’s functionality, adopting best practices ensures efficiency, maintainability, and scalability

- Keep scripts simple: Break complex scripts into smaller, manageable functions or script includes.

- Use comments: Properly document scripts to improve readability and maintainability.

- Reuse code: Use Script Includes for reusable server-side logic, avoiding redundant code.

- Error handling: Implement appropriate error handling to avoid unexpected failures.

- Test thoroughly: Test your scripts across different environments to ensure they work as intended (e.g., dev, test, prod).

Example: Error Handling in Business Rule

```javascript

(function executeRule(current, previous) {

try {

var gr = new GlideRecord('incident');

gr.get(current.sys\_id);

gr.short\_description = 'Record created: ' + current.number;

gr.update();

} catch (e) {

gs.error('Error in Business Rule: ' + e.message);

}

})(current, previous);

**6. Advanced Techniques**

Once you’re comfortable with the basics, you can explore advanced techniques for scripting in ServiceNow.

- Script Includes: These are reusable server-side scripts that can be invoked from other server-side scripts, UI actions, or even workflows.

- Custom Business Rules: Beyond basic create/update/delete triggers, you can build more complex business logic for data validation, integrations, etc.

- Scheduled Jobs: Use them to automate routine tasks like data cleanup, report generation, or system monitoring.

Example: Script Include

```javascript

var IncidentUtils = Class.create();

IncidentUtils.prototype = {

initialize: function() {},

closeIncidentsByUser: function(userID) {

var gr = new GlideRecord('incident');

gr.addQuery('assigned\_to', userID);

gr.addQuery('state', '<', 6); // Not closed

gr.query();

while (gr.next()) {

gr.state = 7; // Closed

gr.update();

}

return 'Incidents closed for user ' + userID;

},

type: 'IncidentUtils'

};

**7. Debugging and Performance Optimization**

Knowing how to debug effectively and optimize your scripts is crucial to avoid slow performance or logic errors.

- Debugging Tools: ServiceNow provides several debugging tools:

- JavaScript Log: Useful for debugging client-side scripts.

- Script Debugger: Allows stepping through server-side scripts during execution.

- gs.log() and gs.error(): Log information to the system logs for analysis.

- Performance Considerations:

- Minimize the number of database queries (use GlideAggregate for aggregation).

- Avoid global GlideRecord queries in loops.

- Always test for optimal performance and minimize the load on the system.

**8. Learn from the Community and Documentation**

ServiceNow has a strong community and a wealth of resources:

- ServiceNow Developer Documentation: The official documentation provides examples, API references, and best practices.

- Community and Developer Forums: Engage with other developers on the ServiceNow community and developer forums to learn new techniques, ask questions, and share knowledge.

- ServiceNow Developer Program: Enroll in the developer program to get access to a personal development instance for hands-on practice.

**3) Gain experience in scripting through hands-on exercises and examples.**

Ans: To gain experience in ServiceNow scripting and improve your skills, it's essential to work through hands-on exercises that mirror real-world scenarios. Below are several practical exercises, examples, and tips to help you develop proficiency with both client-side and server-side scripting. Each section progressively builds on key areas, offering experience in essential ServiceNow development.

**Exercise 1**: Create a Simple Client Script

Client scripts allow you to control how forms behave. This first exercise will help you write a client script to auto-populate a field based on a user's selection.

**Objective** :- Auto-fill the Short Description field when the user selects a category on the Incident form.

**Steps**:

1. Navigate to Incident > Open and select a record.

2. Go to Configure > Client Scripts, then click New to create a new client script.

3. Set the following properties:

- Name: Populate Short Description

- Table: Incident

- Type: onChange

- Field Name: Category

- Condition: None

4. Add the following script:

```javascript

function onChange(control, oldValue, newValue, isLoading, isTemplate) {

if (isLoading || newValue === '') {

return;

}

// Automatically fill Short Description based on Category

if (newValue == 'network') {

g\_form.setValue('short\_description', 'Network Issue');

} else if (newValue == 'hardware') {

g\_form.setValue('short\_description', 'Hardware Issue');

} else {

g\_form.clearValue('short\_description');

}

}

5. Save the script and test by changing the Category field in an Incident record.

Exercise 2: Server-side GlideRecord Operations

In this exercise, you will create a Business Rule to perform a database query and update records based on certain conditions.

**Objective**:- Automatically resolve incidents that are in a specific state and have not been updated for 30 days.

**Steps**:

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

2. Set the following properties:

- Name: Auto-resolve Old Incidents

- Table: Incident

- When to run: Before

- Insert condition: State is not resolved and Last Updated is more than 30 days ago.

3. Add the following script:

```javascript

(function executeRule(current, previous) {

var gr = new GlideRecord('incident');

gr.addQuery('state', '!=', 6); // Not Resolved

gr.addEncodedQuery('sys\_updated\_onRELATIVELE@dayofweek@ago@30'); // Not updated in 30 days

gr.query();

while (gr.next()) {

gr.state = 6; // Resolved

gr.work\_notes = 'Auto-resolved after 30 days of inactivity';

gr.update();

}

})(current, previous);

4. Save and test the Business Rule by creating test incidents that meet the criteria. Exercise 3: GlideAjax for Client-Server Communication

In this exercise, you’ll create a GlideAjax script that allows client scripts to call a server-side function.

**Objective**:- Fetch additional data from the server when a user selects an incident priority.

**Steps**:

1. Create a Script Include (Server-side):

- Navigate to System Definition > Script Includes and click New.

- Name: FetchPriorityDetails

- API Name: FetchPriorityDetails

- Set Accessible from: Client Callable

- Add the following script:

```javascript

var FetchPriorityDetails = Class.create();

FetchPriorityDetails.prototype = Object.extendsObject(AbstractAjaxProcessor, {

getPriorityDetails: function() {

var priority = this.getParameter('sysparm\_priority');

// Logic to return details based on priority

if (priority == '1') {

return 'High priority incident, requires immediate attention.';

} else if (priority == '2') {

return 'Medium priority, attend within 24 hours.';

} else {

return 'Low priority, resolve as time permits.';

}

}

});

- Save the Script Include.

2. Create a Client Script (Client-side):

- Navigate to Incident > Open, select a record, and go to Configure > Client Scripts.

- Name: Fetch Priority Details

- Table: Incident

- Type: onChange

- Field Name: Priority

- Add the following client script:

```javascript

function onChange(control, oldValue, newValue, isLoading, isTemplate) {

if (isLoading || newValue === '') {

return;

}

var ga = new GlideAjax('FetchPriorityDetails');

ga.addParam('sysparm\_name', 'getPriorityDetails');

ga.addParam('sysparm\_priority', newValue);

ga.getXMLAnswer(function(response) {

var priorityMessage = response;

g\_form.setValue('description', priorityMessage);

});

}

3. Save and test by changing the Priority field on an Incident form.

Exercise 4: Script Include for Reusable Code

You will create a Script Include to centralize reusable logic, which can be called from various places like UI Actions, Business Rules, or other scripts.

**Objective**:- Create a Script Include to handle task assignments based on the group the user belongs to.

**Steps**:

1. Create the Script Include:

- Navigate to System Definition > Script Includes and click New.

- Name: TaskAssignmentUtil

- API Name: TaskAssignmentUtil

- Accessible from: All Application Scopes

- Script:

```javascript

var TaskAssignmentUtil = Class.create();

TaskAssignmentUtil.prototype = {

initialize: function() {},

assignToGroupMember: function(task, groupID) {

var gr = new GlideRecord('sys\_user');

gr.addQuery('sys\_user\_group', groupID);

gr.query();

if (gr.next()) {

task.assigned\_to = gr.sys\_id;

task.update();

return 'Task assigned to ' + gr.name;

} else {

return 'No members found in group';

}

}

};

- Save the Script Include.

2. Call the Script Include from a UI Action:

- Navigate to System Definition > UI Actions and click New.

- Name: Assign to Group Member

- Table: Incident

- Action name: assign\_to\_group\_member

- Script:

```javascript

var taskUtil = new TaskAssignmentUtil();

var result = taskUtil.assignToGroupMember(current, 'YOUR\_GROUP\_ID');

gs.addInfoMessage(result);

```

- Save and test by selecting this action from an Incident.

Exercise 5: Debugging and Error Handling

Debugging is crucial for identifying issues. In this exercise, you’ll practice using logging and error handling.

**Objective**:- Create a Business Rule that logs error messages if something fails.

**Steps**:

1. Navigate to System Definition > Business Rules and create a new rule.

2. Set the following properties:

- Name: Logging Example

- Table: Incident

- When: Before Insert

3. Add the following script:

```javascript

(function executeRule(current, previous) {

try {

// Simulate an error

var gr = new GlideRecord('invalid\_table');

gr.query();

} catch (e) {

gs.error('Error occurred: ' + e.message);

}

})(current, previous);

```

4. Save and test by creating a new incident to trigger the error.

**4) Develop the skills necessary to create efficient and effective scripts for ServiceNow**

**applications.**

**Ans**: By focusing on scripting for ServiceNow, you'll be able to develop and enhance functionality within the platform, creating custom solutions to meet specific business needs. The key areas you can expect to focus on include:

**1. Client-Side Scripting**: This involves scripts that run in the user’s browser, typically used to control forms and the UI in real-time. You'll learn how to use UI policies, client scripts, and DOM manipulation to improve user interactions.

**2. Server-Side Scripting**: This refers to scripts executed on the ServiceNow server, typically used for back-end processes such as querying the database, executing business logic, or performing complex workflows. It includes using business rules, script includes, and scheduled jobs.

**3. Glide API:** You will frequently use the Glide API, which allows you to interact with ServiceNow’s database and perform operations such as querying records, updating records, and more.

**4. ACLs and Security**: Writing scripts to control access to records or fields, ensuring that users only see or modify what they're authorized to.

**5. Integration Scripts**: Scripting for integrating ServiceNow with external systems, handling API requests, and responses using formats like JSON and XML.

By focusing on these core concepts, hands-on practice, and working on real-world scenarios, you'll gain a solid foundation to script effectively in ServiceNow.

**5) Enhance problem-solving abilities by applying scripting techniques to real-world scenarios in ServiceNow.**

**Ans**: Enhancing problem-solving abilities through scripting in ServiceNow involves using your knowledge of both client-side and server-side scripting to address real-world challenges. In ServiceNow, scripting can be used to automate tasks, enhance user experiences, and customize functionalities beyond what’s available out of the box. By mastering scripting techniques, you can develop more efficient workflows and solve complex issues within the platform.

Key Areas to Focus on:

**1. Client-Side Scripting (Front-end Enhancements):**

- Client Scripts: You can use client scripts to control UI elements or validate user inputs dynamically. For example, hiding fields based on user selections or pre-filling form fields based on other data.

- UI Policies: These allow for dynamic changes to forms based on conditions without needing deep coding skills.

- UI Actions: These scripts allow you to add buttons, links, and context menu items to forms and lists.

**2. Server-Side Scripting (Back-end Automations):**

- Business Rules: Server-side scripts like business rules help automate tasks in the background, such as setting values, enforcing validation, or triggering events based on changes in data.

- Script Includes: These allow for reusable server-side code that can be called from other scripts, improving efficiency and reducing redundancy.

- ACL Scripting: Controls security and access by determining who can read, write, or create records.

**3. Integrations and Data Handling:**

- GlideRecord: A powerful class used to query, update, or delete records in ServiceNow.

- REST and SOAP Integrations: Through script includes and business rules, you can integrate ServiceNow with external systems.

- Transform Maps & Fix Scripts: You can write scripts to transform incoming data during imports or modify records at the database level.

**4. Advanced Problem-Solving Techniques:**

- Debugging: Using tools like the Script Debugger or system logs to troubleshoot and optimize your code.

- Real-time Use Cases: Writing scripts for Service Portal widgets, email notifications, and automation within workflows.

**Benefits of Scripting in Real-World Scenarios:**

- Customization: Tailor the platform to your organization's specific needs, enhancing user experiences and optimizing workflows.

- Automation: Reduce manual tasks by automating processes such as approval workflows, notifications, and data validation.

- Scalability: Create reusable components like script includes and business rules to build scalable solutions.

- Efficiency: Using efficient scripts improves the overall performance of the system, reducing lag and errors.