

ServiceNow 201:

3 - Scripting Locations



Course Outline

1 Course Introduction

2 Development Overview

3 Scripting Locations

4 GlideRecord

5 GlideSystem

6 GlideForm & GlideUser

7 GlideAjax

8 Exploring Other APIs

9 Becoming A Scripting Master

10 Creating A Custom App

Section Outline

1 Introduction

2 Scripting Locations

3 Business Rules

4 Client Scripts

5 UI Actions

6 UI Policies

7 Script Includes

8 Scheduled Jobs

9 Workflow Scripting

10 Where To Customize?

11 APIs

12 Where Can I Use This?

Scripting Locations



Business Rules



Script Includes



Client Scripts



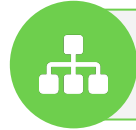
UI Actions



UI Policies



Scheduled Jobs



Workflows



Transform Maps



Web Services



UI Pages & UI Macros



Service Portal Widgets



And more

Business Rules



- Server-side
- Most common scripting location
 - JavaScript that runs on server
- Triggered by database operations
- ~2,400 out-of-box business rules
- Access to:
 - Current object (current)
 - Previous object (previous)
 - Scratchpad (g_scratchpad)



A business rule is a server-side script that runs when a record is displayed, inserted, updated, or deleted, or when a table is queried.

[ServiceNow Docs](#)



Form View: Business Rules

- Runs on a specific table
- 2 execution conditions
 - Database operation
 - Custom condition
- When to run

The screenshot shows the 'Business Rule' configuration page for 'Create Asset on insert'. The interface includes a header with navigation icons and buttons for 'Update', 'Save', 'Delete', and sort arrows. A blue informational box explains that a business rule is a server-side script that runs on specific events. The main configuration area has tabs for 'When to run', 'Actions', and 'Advanced'. The 'When to run' tab is active, showing fields for 'Name' (Create Asset on insert), 'Table' (Configuration Item [cmdb.]), 'Application' (Global), and 'Active' (checked). Below these are 'When' (before) and 'Order' (1,000) settings. The 'Advanced' tab is also visible, showing a 'Condition' field with the script `current.asset.nil() || (current.asset.ci != current.sys_id)` and a 'Script' field with the code `var ca = new AssetandCI();` and `ca.createAsset(current);`. A green callout box highlights the 'Advanced' tab and the 'Condition' and 'Script' fields. Another green callout box points to the 'When' and 'Order' fields. A third green callout box points to the 'Filter Conditions' section, which includes buttons for 'Add Filter Condition' and 'Add "OR" Clause', and a section for 'Role conditions' with an edit icon.

Business Rule: Create Asset on insert

A business rule is a server-side script that runs when a record is displayed, inserted, deleted, or when a table is queried. Use business rules to automatically change values in form fields when the specified conditions are met. [More Info](#)

Name: Create Asset on insert

Table: Configuration Item [cmdb.]

Application: Global

Active: ☒

When to run: before

Order: 1,000

Filter Conditions: Add Filter Condition Add "OR" Clause

Role conditions:

Condition: `current.asset.nil() || (current.asset.ci != current.sys_id)`

Script:

```
1 var ca = new AssetandCI();
2 ca.createAsset(current);
3
4
5
```

Update: ☐

Delete: ☐

Query: ☐

Example: Business Rules

1. User sends request to server for specific incident (query)
2. Application server requests record from database server
3. Database server responds to application server with record
4. Application server checks for display business rules, then sends the response back to the user
5. User modifies incident record and sends update request
6. Application server receives update, checks for before business rules, then sends to database server
7. Database server updates record
8. Application server checks for after business rules



Use Cases: Business Rules

Use Case #1

- Create an associated **CI** when a new **asset** is *created*

Use Case #2

- When an **incident** is *reopened*, *increment* the **reopen count**

Client Scripts

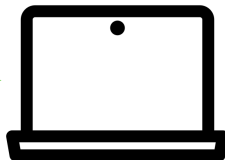


- JavaScript on client-side
- Form views
- Triggered by
 - Field changes
 - Page loads
 - Form submissions
 - Cell edits
- Shipped to the browser



Client scripts run on the client (web browser). You can use client scripts to define custom behaviors that run when events occur such as when a form is loaded or submitted, or a cell changes value.





[ServiceNow Docs](#)
















Form View: Client Scripts

- Table
- UI Type
- Type
- Field name
- Script

< ☰ Client Script Highlight VIP Caller



Update Save Delete

Name	Highlight VIP Caller	Application	Global	
Table	Incident [incident]	Active	<input checked="" type="checkbox"/>	
UI Type	Desktop	Inherited	<input type="checkbox"/>	
Type	onChange	Global	<input checked="" type="checkbox"/>	
Field name	Caller			
Description	<div></div>			
Messages	<div></div>			
Script	<div></div> <pre>1 function onChange(control, oldValue, newValue, isLoading) { 2 var callerLabel = \$('label.incident.caller_id'); 3 var callerField = \$('sys_display.incident.caller_id'); 4 if (!callerLabel !callerField) 5 return; 6 7 if (!newValue) { 8 callerLabel.setStyle({backgroundImage: ""}); 9 callerField.setStyle({color: ""}); 10 return; 11 } 12 g_form.getReference('caller_id', vipCallerCallback);</pre>			

Use Cases: Client Scripts

Use Case #1

- *Highlight* **Caller** field if user is a **VIP**

Use Case #2

- *Run* **conflict checker** for Change Management

UI Actions



- Server-side or client-side
- UI actions can be
 - Buttons
 - Menu items
 - Links
- Typically configured for form views



UI actions add buttons, links, and context menu items on forms and lists, making the UI more interactive, customizable, and specific to user activities. UI actions can contain scripts that define custom functionality.

[ServiceNow Docs](#)



The screenshot shows a 'Change Request' form for 'CHG0030006'. The form fields include 'Number', 'Requested by', 'Category', 'Configuration Item', and 'Priority'. A context menu is open over the 'Category' field, which is set to 'Other'. The menu options are: Save, Add to Visual Task Board, Close Change, Refresh Impacted Services, Metrics Timeline, Follow on Live Feed, Show Live Feed, Edit Risk Conditions, and Configure. A green arrow points to the 'Category' field.

A row of four buttons: 'Update', 'Save', 'Copy Change', and 'Delete'.

Related Links

[Calculate Risk](#)
[Show Workflow](#)
[Workflow Context](#)

Form View: UI Actions

- Table
- Show on insert/update
- Client
- Type of UI action
- Condition
- Script

UI Action
Create Change

Name: Create Change

Application: Global

Table: Problem [problem]

Order: 100

Action name:

Active: ☒

Show insert: ☐

Show update: ☒

Client: ☐

Form button: ☐

Form context menu: ☒

Form link: ☐

List banner button: ☐

List bottom button: ☐

List context menu: ☐

List choice: ☐

List link: ☐

Overrides:

Comments:

Hint:

Condition: gs.hasRole("itil") && current.rfc.nil()

Script:

```
1 var change = new GlideRecord("change_request");
2 change.short_description = current.short_description;
3 change.description = current.description;
4 change.cmdb_ci = current.cmdb_ci;
5 change.priority = current.priority;
6 change.company = current.company;
7 change.sys_domain = current.sys_domain;
8 var sysID = change.insert();
```

Server-Side UI Actions



- Access to:
 - current object
 - GlideRecord API
 - GlideSystem API
- Default behavior
- Runs server-side if **Client** checkbox is not checked
- Better performance

Name	<input type="text" value="Submit"/>
Table	Dashboard [pa_dashboards] ▼
Order	<input type="text" value="-100"/>
Action name	<input type="text" value="sysverb_insert"/>
Active	<input checked="" type="checkbox"/>
Show insert	<input checked="" type="checkbox"/>
Show update	<input type="checkbox"/>
Client	<input type="checkbox"/>

Client-Side UI Actions



- Check the Client checkbox
- Provide the name of the Onclick method
- Use `action.setRedirectURL()` to redirect user to another page
- To call another UI action on the current form, use one of the following:

- `gsftSubmit(gel(<ui_action_name>))`
- `gsftSubmit(null, g_form.getFormElement(), <ui_action_name>)`

The screenshot shows the configuration for a UI Action named 'Resolve Incident'. The fields and their values are:

- Name: Resolve Incident
- Table: Incident [incident]
- Order: 100
- Action name: resolve_incident
- Active: ☒
- Show insert: ☒
- Show update: ☒
- Client: ☒ (This field is connected by a green line from the 'Check the Client checkbox' bullet point)
- List v2 Compatible: ☒
- List v3 Compatible: ☐
- Comments: (empty text area)
- Hint: (empty text area)
- Onclick: resolveIncident(); (This field is connected by a green line from the 'Provide the name of the Onclick method' bullet point)
- Condition: (current.incident_state != 7 && current.incident_sta
- Script: (This section contains a code editor with a function definition, connected by a green line from the 'Use action.setRedirectURL()...' bullet point)

```
1 function resolveIncident(){
2   //Set the 'Incident state'
   mandatory fields
```

Server-Side & Client-Side UI Actions



- Server-side & client-side functions
- Must call the UI Action again using
 - `gsftSubmit()`
- Verifies the client-side code is complete
 - `if(typeof window == 'undefined')`

Onclick

`generateVariables()`

```
1 function generateVariables() {
2   var smp = new GlideRecord('sys_soap_message_parameters');
3   smp.addQuery('soap_function', g_form.getUniqueValue());
4   smp.query();
5   if (smp.next()) { //found existing variables that might be deleted
6     var dialog = new GlideDialogWindow("glide_confirm_standard");
7     dialog.setWidth("800");
8     dialog.setTitle(getMessage("Generating Variables"));
9     dialog.setPreference('warning', true);
10    dialog.setPreference('title', getMessage("'Variable substitutions' which are already
11    defined but not referenced in the SOAP envelope will be removed. Do you want to continue?"));
12    dialog.setPreference('onPromptComplete', onPromptComplete);
13    dialog.render(); //Open the dialog
14  } else {
15    onPromptComplete();
16  }
17 }
18
19 function onPromptComplete(){
20   //Call the UI Action again but skip the 'onclick' function
21   gsftSubmit(null, g_form.getFormElement(), 'generate_variables_soap'); //MUST call the
22   'Action name' set in this UI Action
23 }
24
25 //Code that runs without 'onclick'
26 //Ensure call to server-side function with no browser errors
27 if (typeof window == 'undefined')
28   serverResolve();
29
30 function serverResolve(){
31   current.update();
32   var au = new ArrayUtil();
33   var substitutions = f1.
34 }
```


Use Cases: UI Actions

Use Case #1

- *Trigger* Salesforce **integration**, *creating* an associated Salesforce **ticket**

Use Case #2

- *Reject* an **approval** record

UI Policies



- Client-side logic
- Primarily used on forms
- 99% no scripting is required
- Used to set form fields to:
 - Read-only
 - Mandatory
 - Show/hide



UI policies offer an alternative to client scripts for dynamically changing information on a form. Use UI policies to define custom process flows for tasks.

[ServiceNow Docs](#)



Form View: UI Policies

- Table
- Conditions
- UI Policy Actions
- Script

Table

Asset [alm_asset]

Application

Global

Active

Short description

Pre-allocated constraints

Order

90

When to Apply

Script

Conditions

Add Filter Condition

Add "OR" Clause

All of these conditions must be met

Substate

is

Pre-allocated

AND

OR

X

Class

is not

Consumable

AND

OR

X

Class

is not

Software License

AND

OR

X

Global

On load

Reverse if false

Inherit

When to Apply

Script

Run scripts

Run scripts in UI type

Desktop

Execute if true

```
1 function onCondition() {
2
3 // hide sections irrelevant to pre-allocated
4 var sections = g_form.getSections();
5 for (var i = 2; i < sections.length; i++)
6   sections[i].style.display = 'none';
7
8 // hide all related lists
9 g_form.hideRelatedLists();
10 }
```

Execute if false

```
1 function onCondition() {
2 }
```

Related Links

Default view

UI Policy Actions

New

Search

for text

Search

1 to 9

UI policy = Pre-allocated constraints

	Field name	Mandatory	Visible	Read only
<input type="checkbox"/>	install_status	Leave alone	Leave alone	True
<input type="checkbox"/>	assigned	Leave alone	False	Leave alone
<input type="checkbox"/>	location	Leave alone	Leave alone	True

Use Cases: UI Policies

Use Case #1

- Set an incident's **Short description** field to *read-only* if the incident state is **Closed**

Use Case #2

- *Hide* an incident's **Resolution notes** field if the state is **Open**

Script Includes



- Server side JavaScript
- Store JavaScript classes & functions/methods
- Reusable code
- Only runs when invoked
- Unique since they can be called from anywhere



Script includes are used to store JavaScript that runs on the server. Create script includes to store JavaScript functions and classes for use by server scripts. Each script include defines either an object class or a function. Script includes run only when called by a server script.

[ServiceNow Docs](#)



- Name
- Client callable
- Script

- Name
- Client callable
- Script

Script Include Characteristics



- 2 types

- Classless

- Script include name = function name
 - Server-side only

- Class

- typically extend another class
 - Can be invoked via server-side or client-side

- Important attributes

- Script Include name
 - type property
 - prototype

The screenshot displays the 'Script Include' configuration interface for two different script includes. The top configuration is for 'getFormUIAction', which is a 'Classless' type. It has a 'Name' of 'getFormUIAction', an 'API Name' of 'global.getFormUIAction', and is 'Accessible from' 'This application scope only'. The 'Script' field contains a JavaScript function definition. The bottom configuration is for 'AgentScheduleAjax', which is a 'Class' type. It has a 'Name' of 'AgentScheduleAjax', an 'API Name' of 'global.AgentScheduleAjax', and is 'Accessible from' 'This application scope only'. The 'Script' field contains a JavaScript class definition that extends 'AbstractAjaxProcessor'.

Script Include: getFormUIAction

Name: getFormUIAction
API Name: global.getFormUIAction
Application: Global
Accessible from: This application scope only
Client callable: ☐
Active: ☒
Description: get available form UI actions for sys_atf_action table

Script:

```
1 function getFormUIAction(tableName) {  
2   if (GlideStringUtil.nil(tableName)) {  
3     return "table=global*form_action=true*active=true";  
4   }  
5   var currentAndParentTables = GlideDBObjectManager.get().getTables(tableName);  
6   var str = currentAndParentTables.toString();  
7   var tables = str.substring(1, str.length() - 1);  
8   tables += ", global";  
9 }
```

Script Include: AgentScheduleAjax

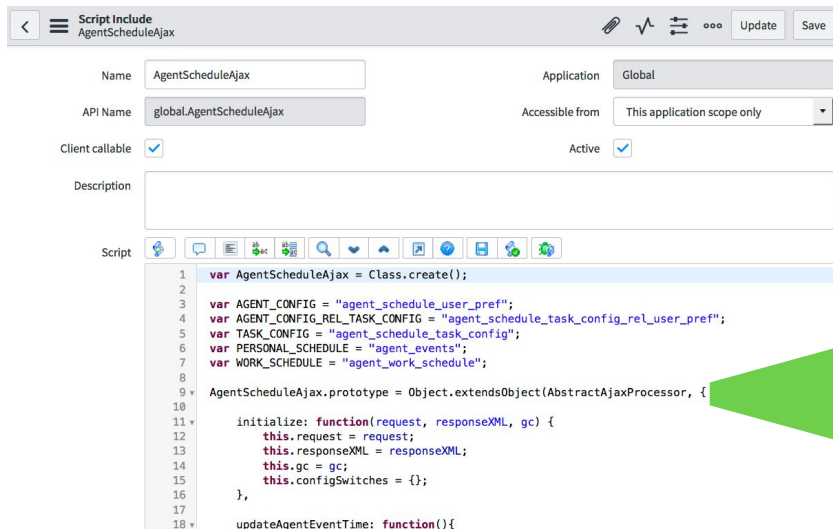
Name: AgentScheduleAjax
API Name: global.AgentScheduleAjax
Application: Global
Accessible from: This application scope only
Client callable: ☒
Active: ☒
Description:

Script:

```
1 var AgentScheduleAjax = Class.create();  
2  
3 var AGENT_CONFIG = "agent_schedule_user_pref";  
4 var AGENT_CONFIG_REL_TASK_CONFIG = "agent_schedule_task_config_rel_user_pref";  
5 var TASK_CONFIG = "agent_schedule_task_config";  
6 var PERSONAL_SCHEDULE = "agent_events";  
7 var WORK_SCHEDULE = "agent_work_schedule";  
8  
9 AgentScheduleAjax.prototype = Object.extendObject(AbstractAjaxProcessor, {  
10  
11   initialize: function(request, responseXML, gc) {  
12     this.request = request;  
13     this.responseXML = responseXML;  
14     this.gc = gc;  
15     this.configSwitches = {};  
16   },  
17  
18   updateAgentEventTime: function() {
```

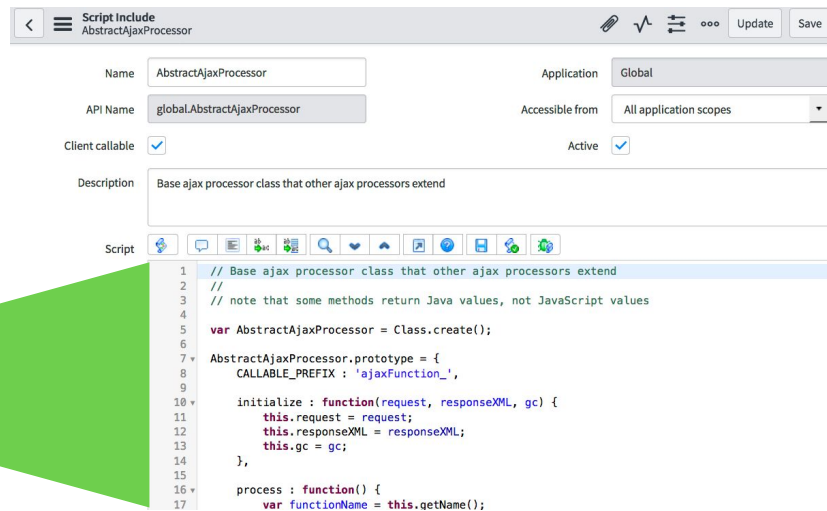
Extending Script Includes

- Any class may be extended
- AbstractAjaxProcessor is a commonly extended class used for GlideAjax
- `<className>.prototype = Object.extendsObject(<extendingClassName>, { /* your script */ });`



The screenshot shows the configuration for a Script Include named 'AgentScheduleAjax'. The 'Name' field is 'AgentScheduleAjax', 'API Name' is 'global.AgentScheduleAjax', and 'Application' is 'Global'. The 'Accessible from' dropdown is set to 'This application scope only'. The 'Client callable' checkbox is checked, and the 'Active' checkbox is also checked. The 'Description' field is empty. The 'Script' area contains the following code:

```
1 var AgentScheduleAjax = Class.create();
2
3 var AGENT_CONFIG = "agent_schedule_user_pref";
4 var AGENT_CONFIG_REL_TASK_CONFIG = "agent_schedule_task_config_rel_user_pref";
5 var TASK_CONFIG = "agent_schedule_task_config";
6 var PERSONAL_SCHEDULE = "agent_events";
7 var WORK_SCHEDULE = "agent_work_schedule";
8
9 AgentScheduleAjax.prototype = Object.extendsObject(AbstractAjaxProcessor, {
10
11   initialize: function(request, responseXML, gc) {
12     this.request = request;
13     this.responseXML = responseXML;
14     this.gc = gc;
15     this.configSwitches = {};
16   },
17
18   updateAgentEventTime: function(){
```



The screenshot shows the configuration for a Script Include named 'AbstractAjaxProcessor'. The 'Name' field is 'AbstractAjaxProcessor', 'API Name' is 'global.AbstractAjaxProcessor', and 'Application' is 'Global'. The 'Accessible from' dropdown is set to 'All application scopes'. The 'Client callable' checkbox is checked, and the 'Active' checkbox is also checked. The 'Description' field contains the text 'Base ajax processor class that other ajax processors extend'. The 'Script' area contains the following code:

```
1 // Base ajax processor class that other ajax processors extend
2 //
3 // note that some methods return Java values, not JavaScript values
4
5 var AbstractAjaxProcessor = Class.create();
6
7 AbstractAjaxProcessor.prototype = {
8   CALLABLE_PREFIX : 'ajaxFunction_',
9
10  initialize : function(request, responseXML, gc) {
11    this.request = request;
12    this.responseXML = responseXML;
13    this.gc = gc;
14  },
15
16  process : function() {
17    var functionName = this.getName();
```


Use Cases: Script Includes

Use Case #1

- *Create* commonly used helper **functions**

Use Case #2

- *Call* a custom **function** via GlideAjax

Scheduled Jobs



- Server-side JavaScript
- Schedule when to run
- **Execute Now** button for testing



Scheduled Jobs are automated pieces of work that can be performed at either a particular time, or on a recurring schedule.

[ServiceNow Docs](#)



- **Schedule**
 - Reports
 - Scripts
 - Charts
 - Etc.
- **When to run (trigger)**
 - Daily/weekly/monthly
 - Periodically
 - Once
 - On Demand

Automation Creator

What would you like to automate?

- Automate the generation and distribution of a report
- Automatically generate something (a change, an incident, a ci, etc) from a template
- Automatically run a script of your choosing
- Automatically run a script to create a report summary table
- Automate the generation and distribution of a custom chart

Run

Time

Conditional

Run this script

1

Run

On Demand

Conditional

☒

Condition

1

Run

Periodically

* Repeat Interval

Days 00

Hours 00

Starting

2017-06-20 18:51:25



Note: You can trigger scheduled jobs in scripts by using the following method `SncTriggerSynchronizer.executeNow()`




Form View: Scheduled Jobs

- Name
- Active
- Run
- Day
- Time
- Run this script
- Execute Now

<

☰

Scheduled Report Summary Generation
Trend of Open Incidents



Update

Save

Execute Now

Delete

Name

Trend of Open Incidents

Active

☒

Summary

🔍

Run

Monthly

▼

Day

1

▼

Time














Hours

03

00

00

Run this script



```
1 var opened = new SummaryTableWriter('incident', '');
2 opened.setTitle("Trend of Open Incidents");
3 opened.setTrend('opened_at', 'month');
4 current.summary = opened.generate();
5 current.setWorkflow(false);
6 current.update();
```

Update

Save

Execute Now

Delete

Use Cases: Scheduled Jobs

Use Case #1

- *Schedule* a monthly **report**

Use Case #2

- *Schedule* a **script** to *retire* old **records**

Workflows



- Automated sequence of activities
- Server-side JavaScript
- Many locations to script in a workflow
- Different scopes

“

The Workflow Editor is an interface for creating and modifying workflows by arranging and connecting activities to drive processes.

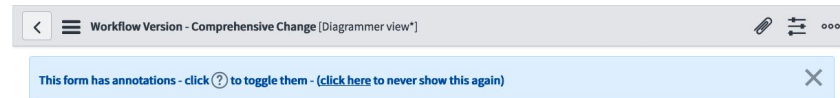
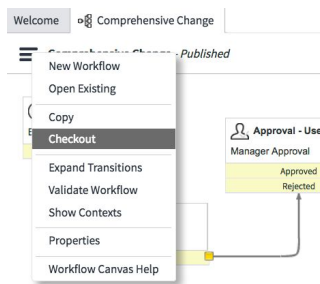
[ServiceNow Docs](#)

”

Workflows



- Workflow contexts are created when workflow conditions evaluate to true
- Versioning and checkouts



Workflow Version - Comprehensive Change [Diagrammer view*]

General Conditions Inputs Activities Application Schedule Estimated Runtime

* Name Comprehensive Change Checked out

* Table Change Request [change_re... Checked out by

Published ☒

Description

General Conditions Inputs Activities Application Schedule Estimated Runtime

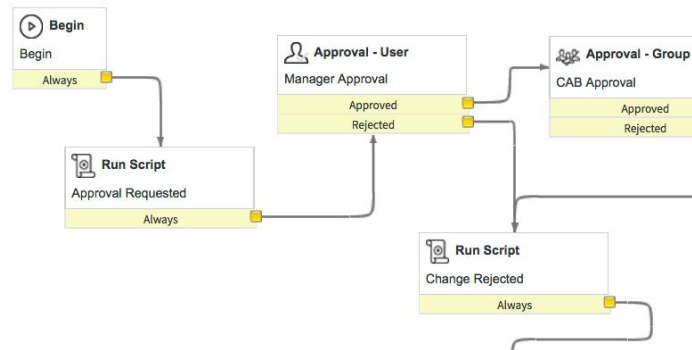
If condition matches Run the workflow Order 10,000

Condition Type is Comprehensive

Scripting in Workflows



- Only use scripts if OOB features aren't enough
- Activities that support scripting
 - Approval activities
 - *Run Script* activity
 - *If*, *Switch*, and *Wait for* condition activities
 - *Create task* activity
 - Notification activities
 - *Scriptable Order Guide* activity
 - *REST Message*, *SOAP Message* activities
- Versioning and checkouts



Name Change Rejected

Stage ?

Script

Script



```
1 current.approval = 'rejected';
```



Dive Deeper: Checkout [this wiki article](#) for a comprehensive guide to all workflow activities

Scripting in Workflows



- Scope

- current record
- workflow.scratchpad object
- Activity specific variables
- Local variables

The screenshot shows the 'Workflow Activity' script editor in 'Diagrammer view'. The script is written in JavaScript and is titled 'Script'. It contains a `run()` function that handles password reset requests. The script includes comments and code for logging, error handling, and updating the workflow's scratchpad with user and process information.

```
1 run();
2
3
4
5 /*
6  * Function determines the reset password activity used in the workflow
7  * based upon the process's credential
8  * store. Function also populates scratch pad variables used throughout the
9  * workflow.
10  * @return void
11  */
12 function run() {
13   var LOGID = '[' + context.name + ' Workflow : ' + activity.name + '
14   Activity] -> ' ;
15   // Retrieve request info:
16   var grRequest = new GlideRecord('pwd_reset_request');
17   if (!grRequest.get(workflow.inputs.u_request_id)) {
18     return logError(LOGID + "The request id " +
19     workflow.inputs.u_request_id + " has no process associated with it.");
20   }
21   workflow.scratchpad.user_sysid = grRequest.user;
22   workflow.scratchpad.process_id = grRequest.process;
23
24   // Retrieve user info:
25   var grSysUser = new GlideRecord("sys_user");
26   if (!grSysUser.get(workflow.scratchpad.user_sysid)) {
27     return logError(LOGID + "The user sysid " +
28     workflow.scratchpad.user_sysid + " is not valid.");
29   }
30 }
```



Note: Use [this wiki article](#) for more information on scripting within workflows

Workflow Activity: *If* condition

- Must set answer to 'yes' or 'no'

Name

Stage

Conditions

Condition (empty)

Advanced ☒

Script

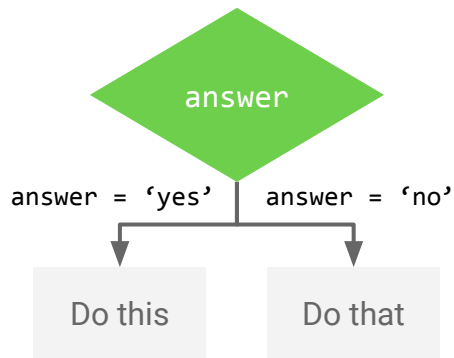


```
1 answer = (function(){
2   if (!JSUtil.nil(workflow.scratchpad.userApprovalIds))
3     return 'yes';
4
5   return 'no';
6 })();
```



The If activity checks a condition or script to determine if a Yes or No transition should be taken. If the workflow creator specifies both the Condition and the Advanced script, both must evaluate successfully for activity to take the Yes transition.


[ServiceNow Docs](#)



Workflow Activity: *Wait for condition*

- Pauses workflow until script evaluates to true
- Wait for condition is evaluated every time the record is updated

Name

Stage 

Specify condition

Condition (empty)

Condition script

```
1 var gr = new GlideRecord("hr_task");
2 gr.addQuery('parent', current.sys_id);
3 gr.addActiveQuery();
4 gr.query();
5 answer = !gr.hasNext();
```



The Wait for condition activity causes the workflow to wait at this activity until the current record matches the specified condition. The workflow evaluates the Wait for condition activity each time the current record is updated. Use this activity to pause a workflow indefinitely until a particular criteria is met by a record update...

[ServiceNow Docs](#)



Workflow Activity: *Approval* - User condition

- Creates a user approval
- answer variable must be set to comma-separated list or array of user or group sys_id's

Advanced ☒

Additional
approvers script

```
1 // Set the variable 'answer' to a comma-separated list of user ids and/or
2 // group ids or an array of user/group ids to add as approvers.
3 // For example:
4 //   answer = [];
5 //   answer.push('id1');
6 //   answer.push('id2');
7
8 if(current.account){
9   answer = [];
10  var gr = new GlideRecord('sys_user_has_role');
11  gr.addQuery('role.name','sn_customerservice.customer_admin');
12  gr.addQuery('user.company',current.account);
13  gr.query();
14  if(gr.getRowCount() == 0)
15    answer = "admin";
16  else {
17    while(gr.next()){
18      var user = new GlideRecord('sys_user');
19      user.get(gr.getValue('user'));
20      answer.push(user.getValue('user_name'));
21    }
22  }
23 }else{
24   answer = "admin";
25 }
26
```

Use Cases: Workflow scripts

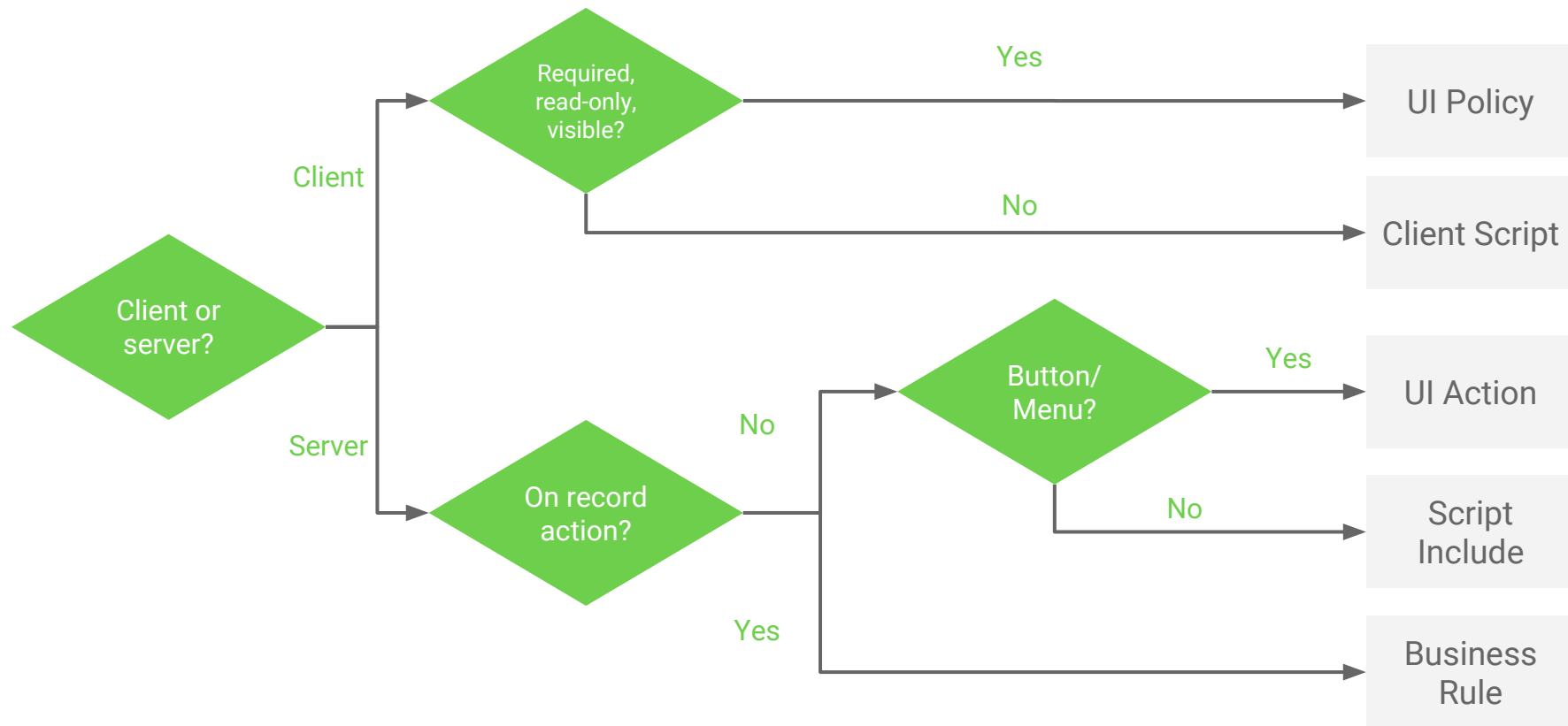
Use Case #1

- Dynamically *assign* **approvals** to a specific group of **users**

Use Case #2

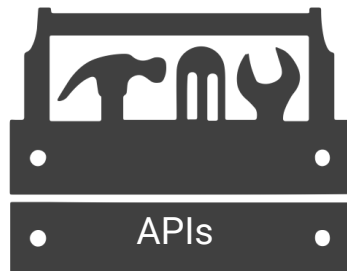
- *Trigger* a web service **call** via a workflow **activity**

Rough Guide To *Where To Customize*



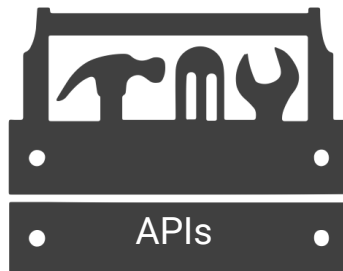
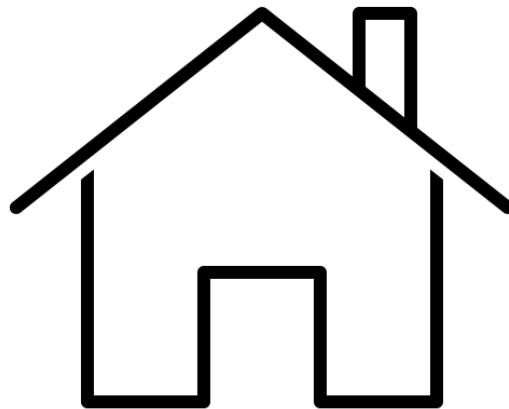
APIs

- **A**pplication **P**rogramming **I**nterface
 - An API provides building blocks to help the developer
 - Can define how to interact with a service or system
 - Common tasks
- **S**erviceNow **A**PIs
 - GlideRecord
 - GlideSystem
 - GlideUser
 - GlideForm
 - GlideAjax
 - GlideDateTime
 - Many more
 - developer.servicenow.com/app.do#!/api_doc



Analogy: Building A House

- Goal: Build a house
- A house needs materials (scripting locations)
 - Foundation - Business Rules
 - Doors - Client Scripts
 - Electrical - UI Actions
 - Materials - Script Includes
 - Etc.
- Tools (APIs)
 - Hammer - GlideRecord
 - Screwdriver - GlideSystem
 - Wrench - GlideUser
 - Etc.



Scripting Locations



Business Rules



Script Includes



Client Scripts



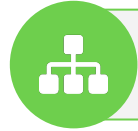
UI Actions



UI Policies



Scheduled Jobs



Workflows



Transform Maps



Web Services



UI Pages & UI Macros



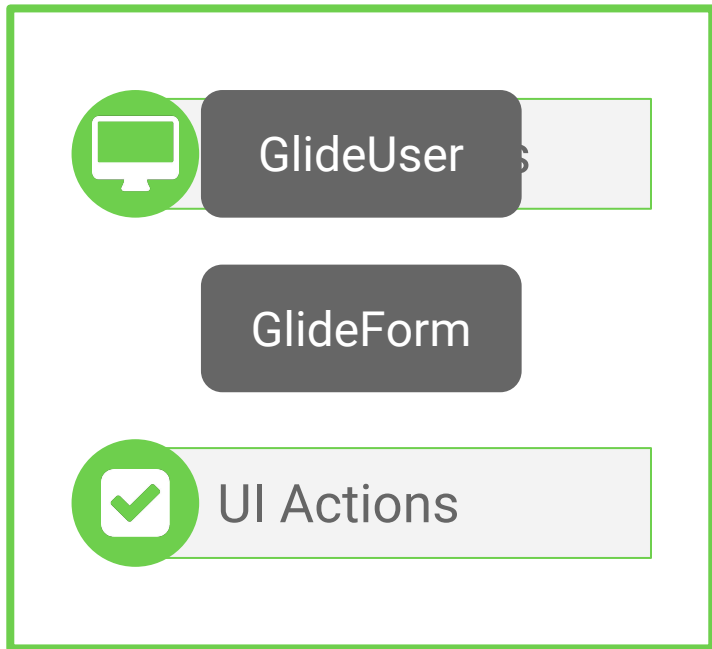
Service Portal Widgets



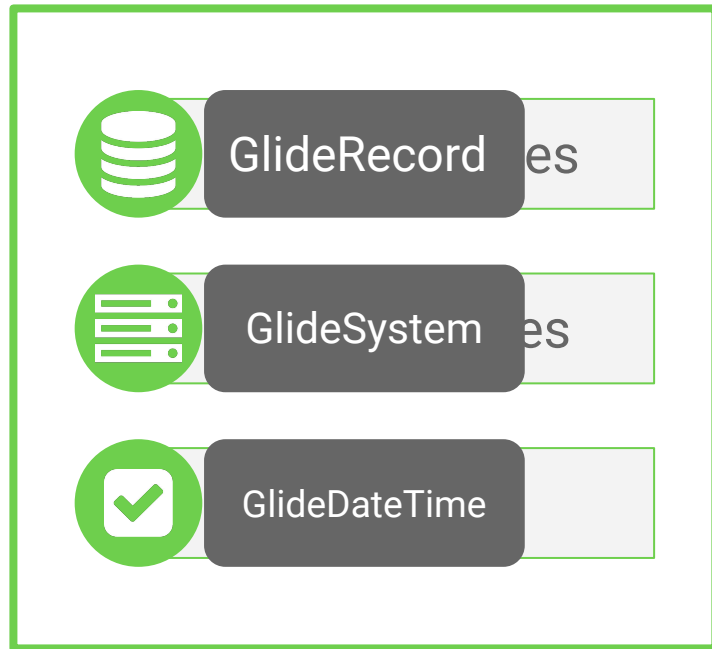
And more

Where Can I Use This?

Client Side

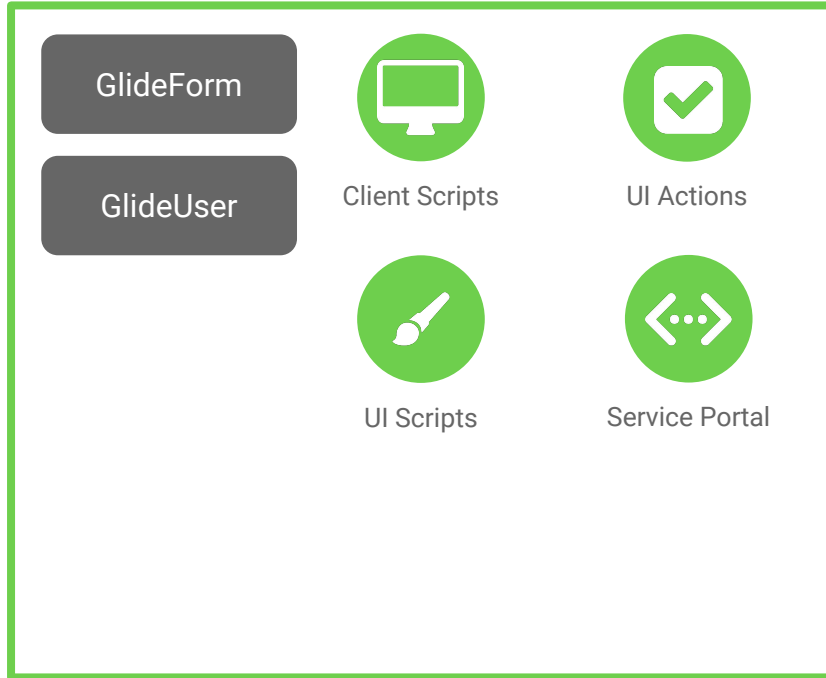


Server Side



Where Can I Use This? (cont.)

Client Side



Server Side

