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

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

Business Rule
Create Asset on insert

Update | Save | Delete

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
Application: Global
Table: Configuration Item [cmdb..
Active: ☑

When to run | Actions | Advanced

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

Script:

```
var ca = new AssetandCI();
ca.createAsset(current);
```

When to run | Actions | Advanced

Specify whether the business rule should run
should run.

When: before
Order: 1,000

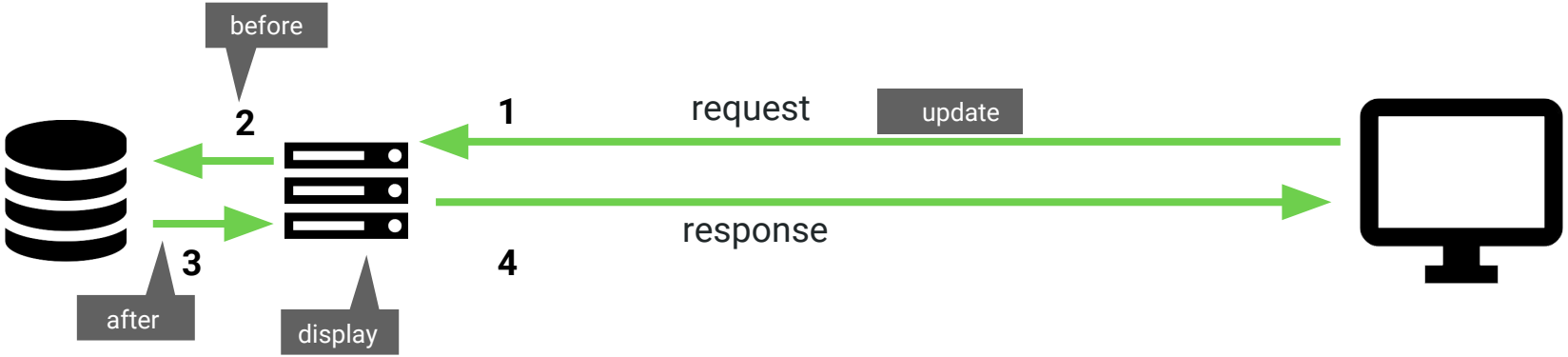
Update ☐
Delete ☐
Query ☐

Filter Conditions: Add Filter Condition | Add "OR" Clause
-- choose field -- | -- oper -- | -- value --

Role conditions

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	☑
UI Type	Desktop	Inherited	☐
Type	onChange	Global	☑
Field name	Caller		
Description			
Messages			

Script

```
function onChange(control, oldValue, newValue, isLoading) {
    var callerLabel = $('label.incident.caller_id');
    var callerField = $('sys_display.incident.caller_id');
    if (!callerLabel || !callerField)
        return;

    if (!newValue) {
        callerLabel.setStyle({backgroundImage: ""});
        callerField.setStyle({color: ""});
        return;
    }
    g_form.getReference('caller_id', vipCallerCallback);
```

Use Cases: Client Scripts

Use Case #1

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

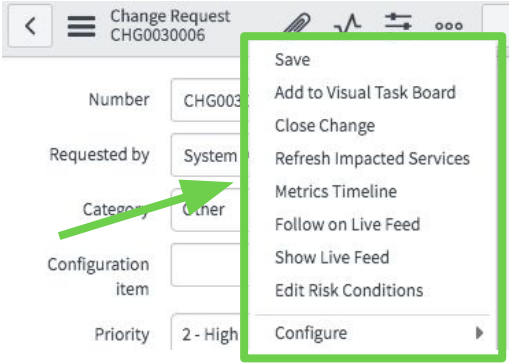
Use Case #2

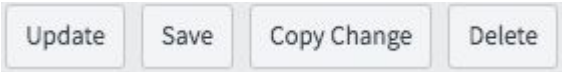
- *Run* **conflict checker** for Change Management

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





Form View: UI Actions

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

UI Action
Create Change

Update | Save | Delete

Name: Create Change
Table: Problem [problem]
Order: 100
Action name:
Active: ☑
Show insert: ☐
Show update: ☑
Client: ☐

Application: Global

Form button: ☐
Form context menu: ☑
Form link: ☐
List banner button: ☐
List bottom button: ☐
List context menu: ☐
List choice: ☐
List link: ☐

Overrides:

Comments:

Hint:

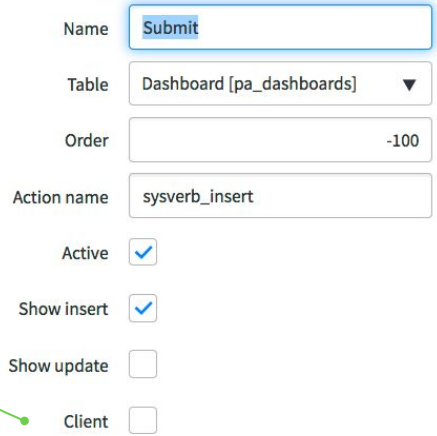
Condition: gs.hasRole("itil") && current.rfc.nil()

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();
```

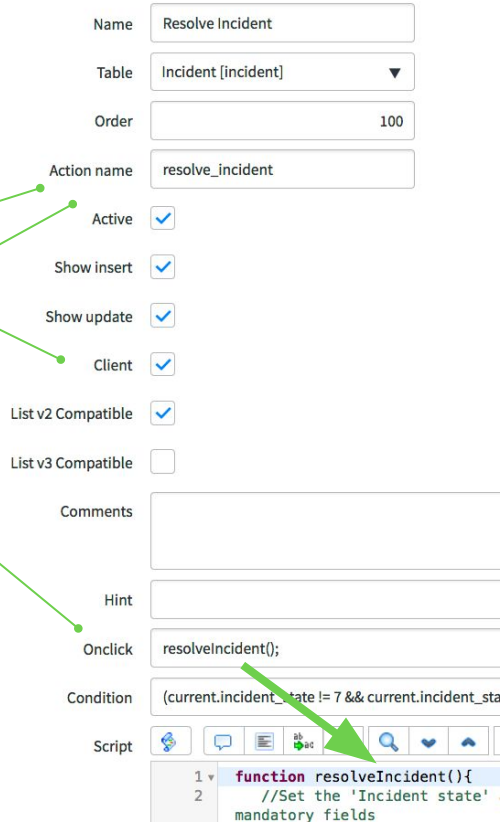
Server-Side UI Actions

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



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>)`



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
   defined but not referenced in the SOAP envelope will be removed. Do you want to continue?"));
11         dialog.setPreference('onPromptComplete', onPromptComplete);
12         dialog.render(); //Open the dialog
13     } else {
14         onPromptComplete();
15     }
16 }
```

```
18 function onPromptComplete(){
19     //Call the UI Action again but skip the 'onclick' function
20     gsftSubmit(null, g_form.getFormElement(), 'generate_variables_soap'); //MUST call the
   'Action name' set in this UI Action
21 }
22
23 //Code that runs without 'onclick'
24 //Ensure call to server-side function with no browser errors
25 if (typeof window == 'undefined')
26     serverResolve();
27
28 function serverResolve(){
29     current.update();
30     var au = new ArrayUtil();
```

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 ✕
Class	is not	Consumable	AND OR ✕
Class	is not	Software License	AND OR ✕

Global: ☑
On load: ☑
Reverse if false: ☐
Inherit: ☑

When to Apply | Script

Run scripts: ☑
Run scripts in UI type: Desktop

Execute if true

```
function onCondition() {

   // hide sections irrelevant to pre-allocated
   var sections = g_form.getSections();
   for (var i = 2; i < sections.length; i++)
     sections[i].style.display = 'none';

   // hide all related lists
   g_form.hideRelatedLists();
}
```

Execute if false

```
function onCondition() {
}
```

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
install_status	Leave alone	Leave alone	True
assigned	Leave alone	False	Leave alone
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

Form View: Script Includes

- Name
- Client callable
- Script

Script Include
SPCart

Update | Save | Delete

Name: SPCart
Application: Global
API Name: global.SPCart
Accessible from: This application scope only
Client callable: ☐
Active: ☑
Description:
Script:

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

// Static methods
SPCart.updateItem = function(itemID, vars) {
    var scCartItemGR = new GlideRecord("sc_cart_item");
    if (!scCartItemGR.get(itemID))
        return false;

    var scItemOptionGR;
    var variable;
    for (var i = 0; i < vars.length; i++) {
        variable = vars[i];
        scItemOptionGR = new GlideRecord("sc_item_option");
        scItemOptionGR.addQuery("cart_item", itemID);
        scItemOptionGR.addQuery("item_option_new.name",
variable.variable_name);
```

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`

Script Include
getFormUIAction

Update Save

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

Script

```
function getFormUIAction(tableName) {
    if (GlideStringUtil.nil(tableName)) {
        return "table=global^form_action=true^active=true";
    }
    var currentAndParentTables = GlideDBObjectManager.get().getTables(tableName);
    var str = currentAndParentTables.toString();
    var tables = str.substring(1, str.length() - 1);
    tables += ", global";
```

Script Include
AgentScheduleAjax

Update Save

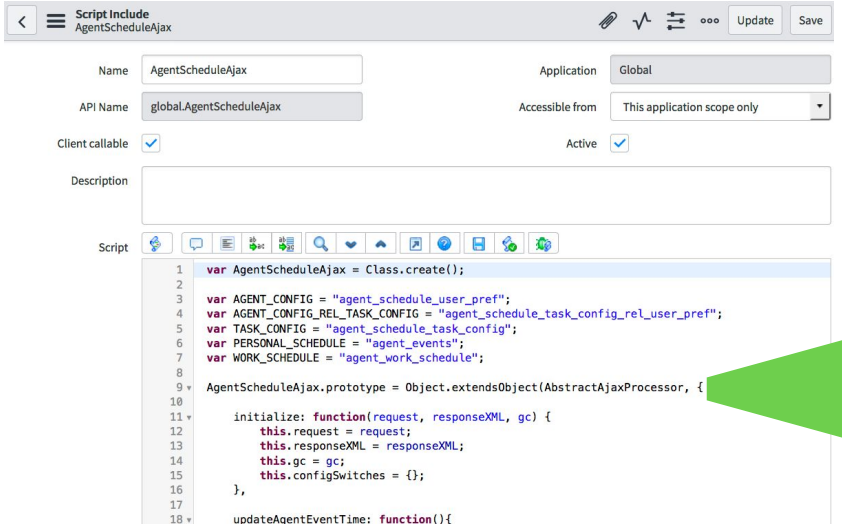
Name: AgentScheduleAjax
Application: Global
API Name: global.AgentScheduleAjax
Accessible from: This application scope only
Client callable: ☑
Active: ☑
Description:

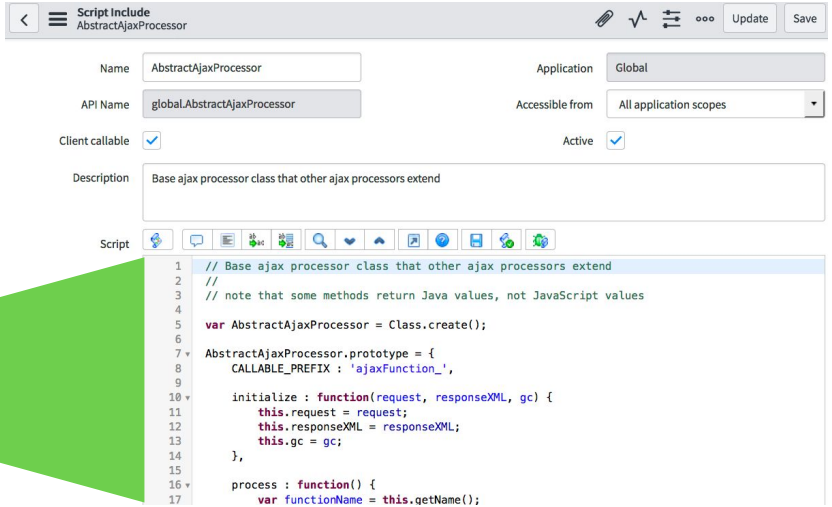
Script

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

var AGENT_CONFIG = "agent_schedule_user_pref";
var AGENT_CONFIG_REL_TASK_CONFIG = "agent_schedule_task_config_rel_user_pref";
var TASK_CONFIG = "agent_schedule_task_config";
var PERSONAL_SCHEDULE = "agent_events";
var WORK_SCHEDULE = "agent_work_schedule";

AgentScheduleAjax.prototype = Object.extendsObject(AbstractAjaxProcessor, {

    initialize: function(request, responseXML, gc) {
        this.request = request;
        this.responseXML = responseXML;
        this.gc = gc;
        this.configSwitches = {};
    },

    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 */ });`





Use Cases: Script Includes

Use Case #1

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

Use Case #2

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

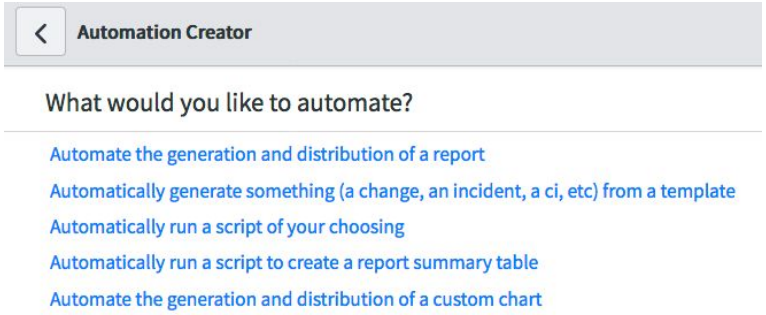
Scheduled Jobs

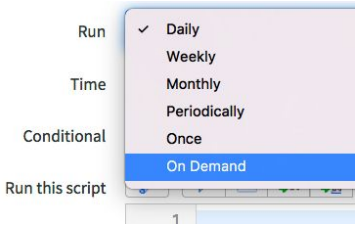
- Server-side JavaScript
- Schedule when to run
- **Execute Now** button for testing

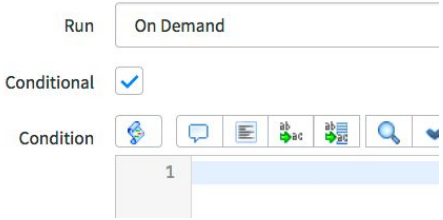
> *Scheduled Jobs are automated pieces of work that can be performed at either a particular time, or on a recurring schedule.*
>
> ServiceNow Docs

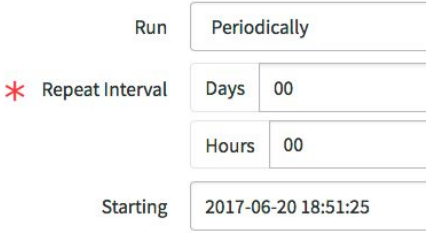
Scheduled Jobs (cont.)

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





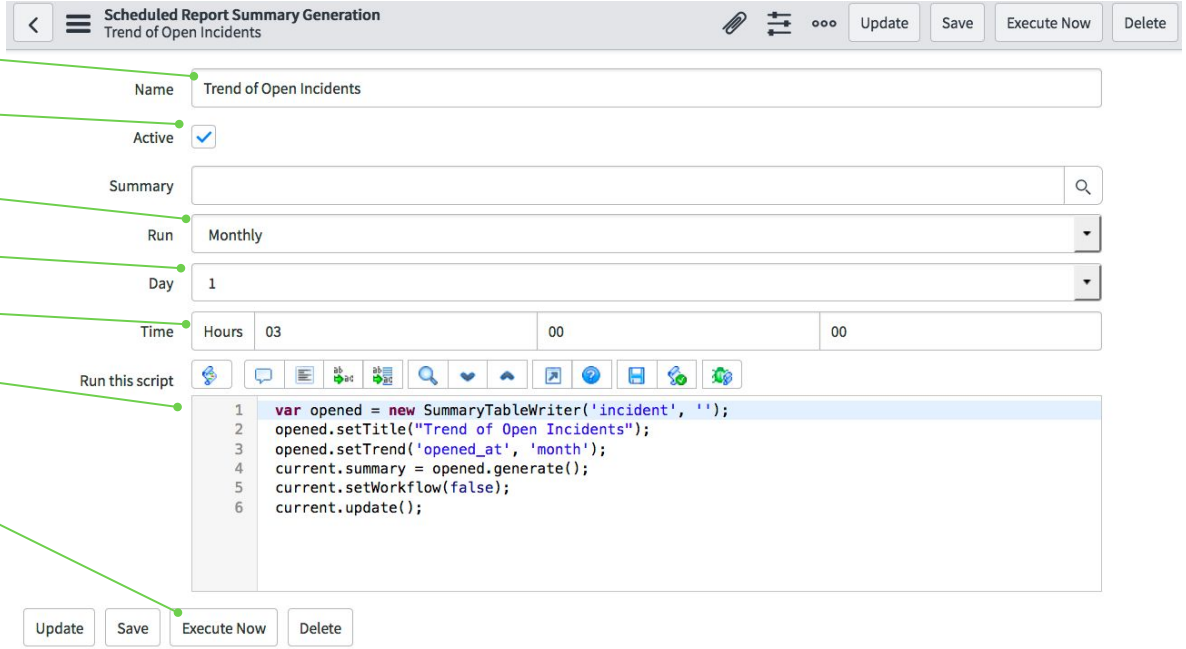




Note: You can trigger scheduled jobs in scripts by using the following method `SncTriggerSynchronizer.executeNow()`

Form View: Scheduled Jobs

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



Use Cases: Scheduled Jobs

Use Case #1

- *Schedule* a monthly **report**

Use Case #2

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

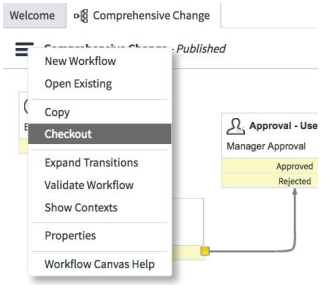
Workflows

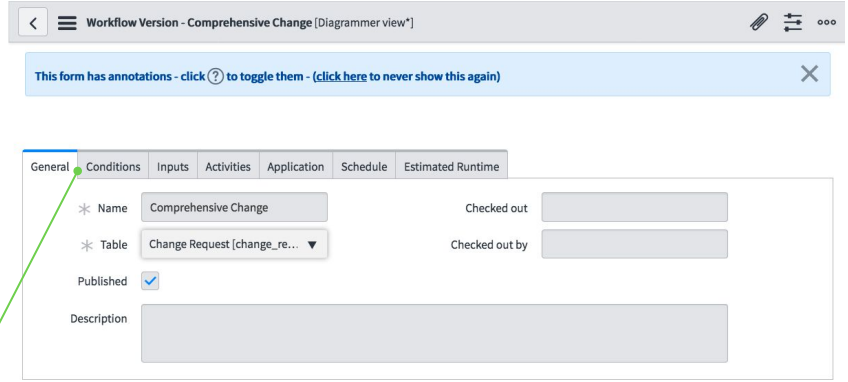
- Automated sequence of activities
- Server-side JavaScript
- Many locations to script in a workflow
- Different scopes

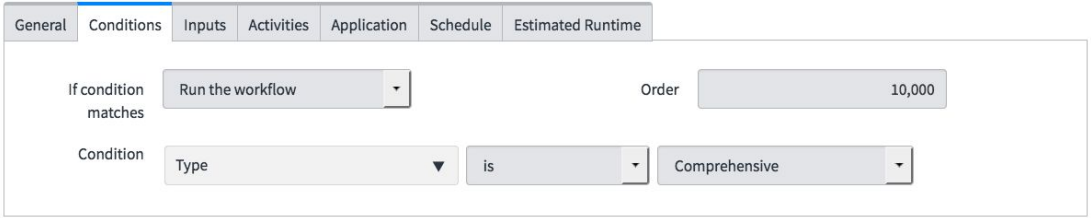
> *The Workflow Editor is an interface for creating and modifying workflows by arranging and connecting activities to drive processes.*
>
> ServiceNow Docs

Workflows

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

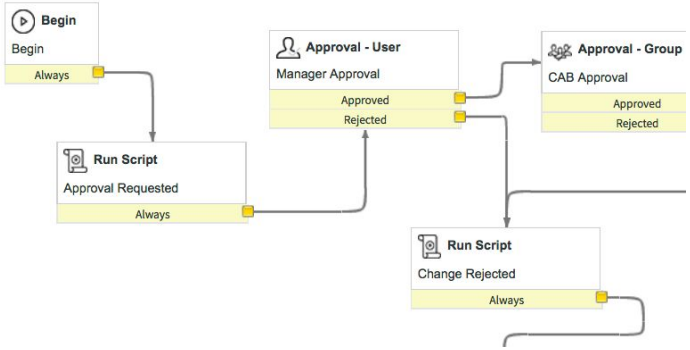


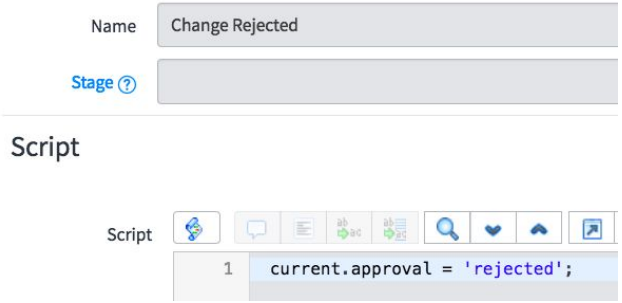




Scripting in Workflows

- 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

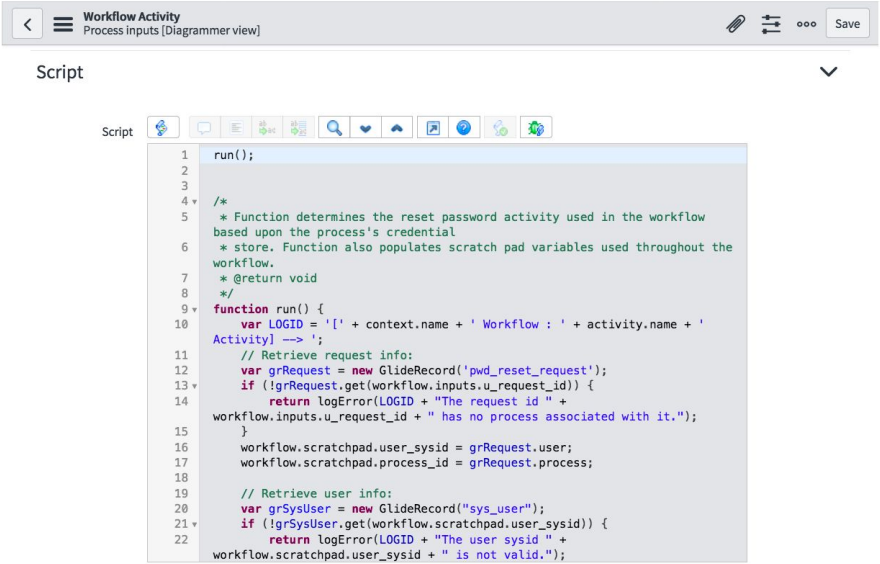




Dive Deeper: Checkout **this wiki article** for a comprehensive guide to all workflow activities

Scripting in Workflows

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



Note: Use this wiki article for more information on scripting within workflows

Workflow Activity: *If* condition

- Must set `answer` to ‘yes’ or ‘no’

Name: Any User approvals?

Stage

Conditions

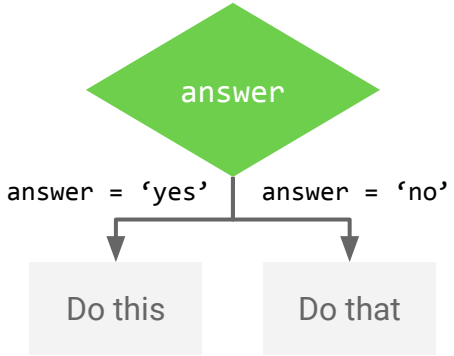
Condition: (empty)

Advanced: ☑

Script

```
answer = (function(){
    if (!JSUtil.nil(workflow.scratchpad.userApprovalIds))
        return 'yes';

    return 'no';
})();
```

> *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: Check all sub-tasks are inactive

Stage

Specify condition

Condition: (empty)

Condition script

```
var gr = new GlideRecord("hr_task");
gr.addQuery('parent', current.sys_id);
gr.addActiveQuery();
gr.query();
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

```
// Set the variable 'answer' to a comma-separated list of user ids and/or
group ids or an array of user/group ids to add as approvers.
//
// For example:
//       answer = [];
//       answer.push('id1');
//       answer.push('id2');

if(current.account){
    answer = [];
    var gr = new GlideRecord('sys_user_has_role');
    gr.addQuery('role.name','sn_customerservice.customer_admin');
    gr.addQuery('user.company',current.account);
    gr.query();
    if(gr.getRowCount() == 0)
        answer = "admin";
    else {
            while(gr.next()){
            var user = new GlideRecord('sys_user');
            user.get(gr.getValue('user'));
            answer.push(user.getValue('user_name'));
        }
    }
}else{
    answer = "admin";
}
```

Use Cases: Workflow scripts

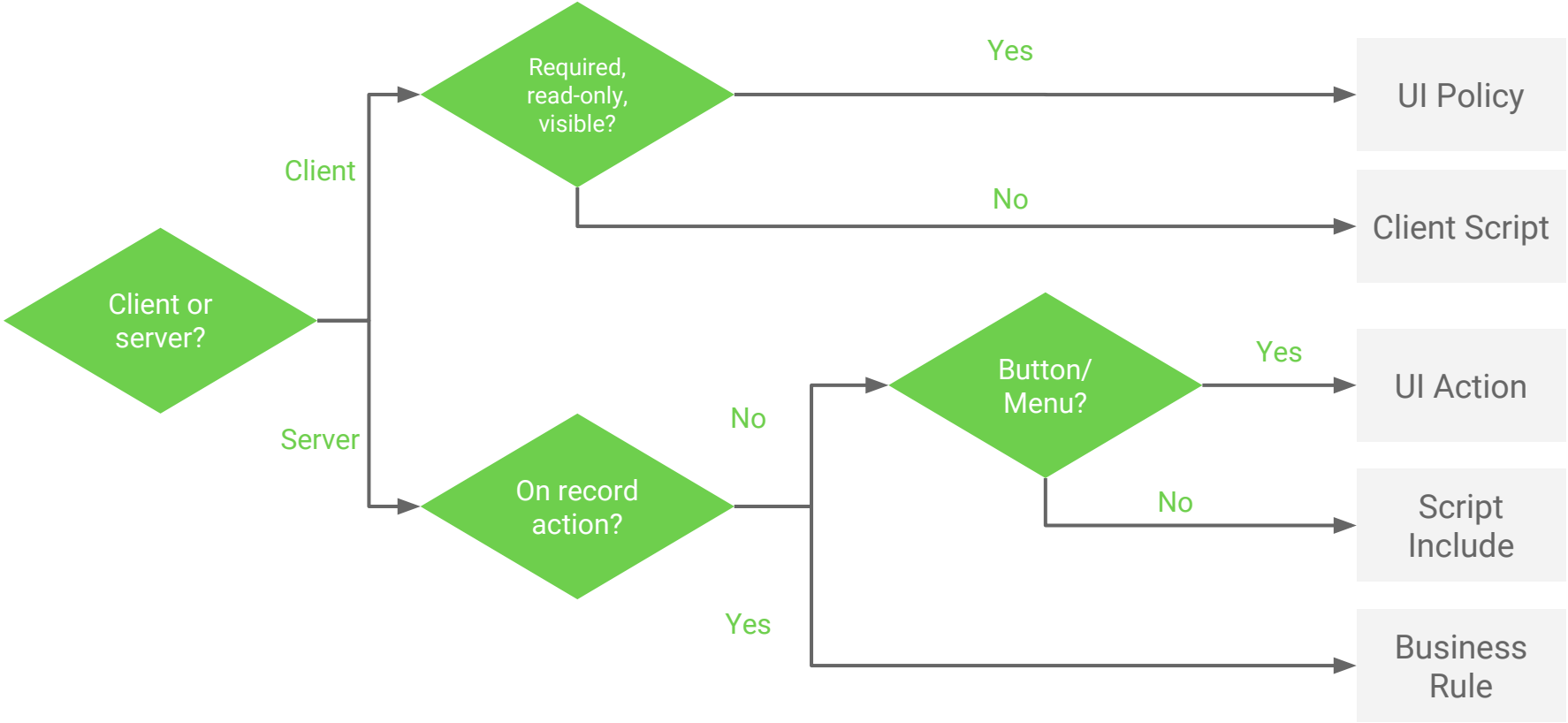
Use Case #1

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

Use Case #2

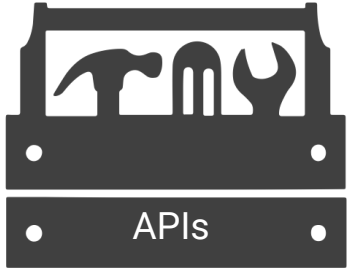
- *Trigger* a web service **call** via a workflow **activity**

Rough Guide To *Where To Customize*



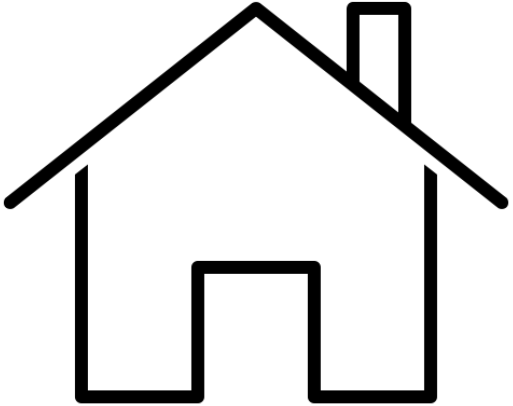
APIs

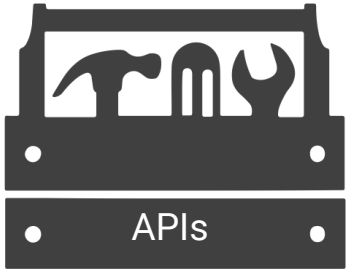
- **A**pplication **P**rogramming **I**nterface
  - An API provides building blocks to help the developer
  - Can define how to interact with a service or system
  - Common tasks
- ServiceNow APIs
  - GlideRecord
  - GlideSystem
  - GlideUser
  - GlideForm
  - GlideAjax
  - GlideDateTime
  - Many more
    - 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.



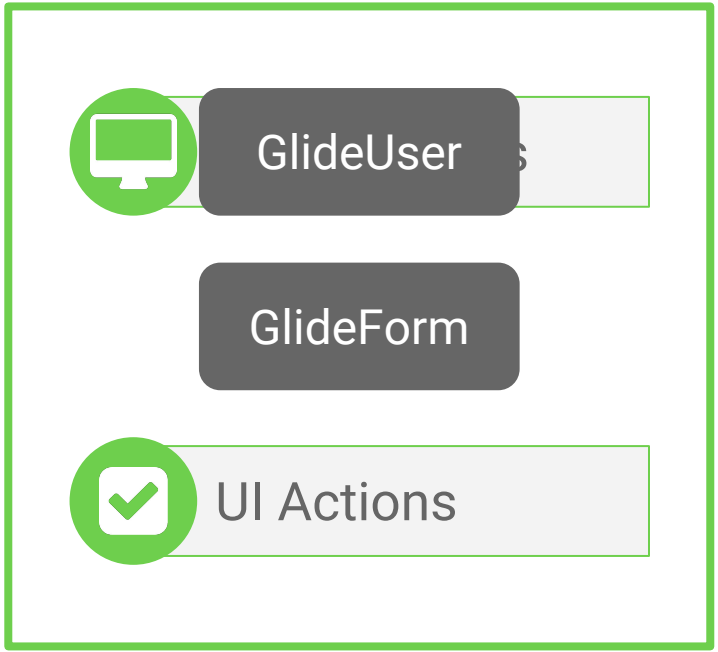


Scripting Locations

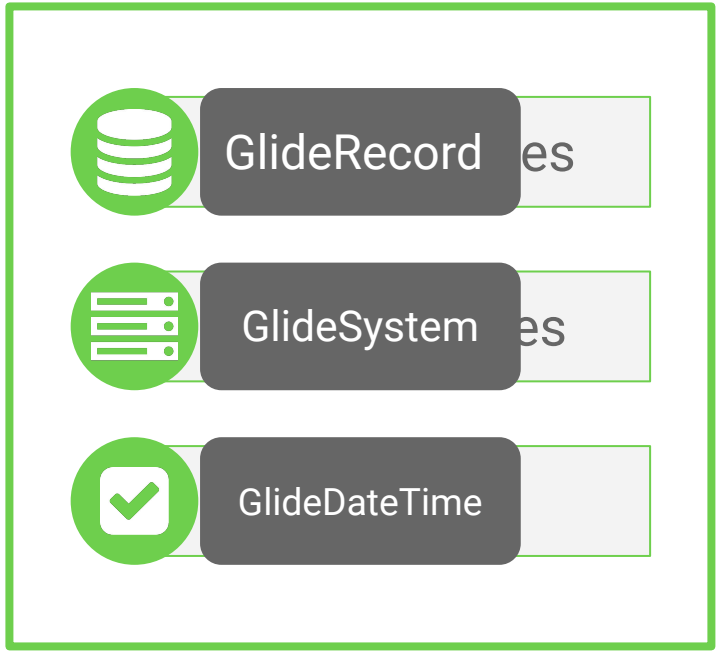
- Business Rules
- Script Includes
- Client Scripts
- UI Actions
- UI Policies
- Scheduled Jobs
- Workflows
- Transform Maps
- Web Services
- UI Pages & UI Macros
- Service Portal Widgets
- And more

Where Can I Use This?

Client Side



Server Side



Where Can I Use This? (cont.)

Client Side

- GlideForm
- GlideUser


Client Scripts


UI Actions


UI Scripts


Service Portal

Server Side

- GlideRecord
- GlideSystem
- GlideDateTime


Business Rules


UI Actions


Script Includes


Scheduled Jobs


Service Portal


Web Services


Workflows