Add Custom Actions

Last Modified on 12/02/2024 1:37 am EST

V10.6

Overview

As a workflow developer, you can now add custom actions relevant to case management in a solution. Addition of custom actions helps ease the solution customization per customer needs.

Below is a comparison between the custom action types:

Custom action type	Frontend	Backend
Use case	An action that requires input from the user via forms. For example: Update case type and Insert comment.	An action that can be executed on a case without additional input from the user. For example: Close case and Mark for QC.
Command	OpenWorkflowWindowCommand	InvokeWorkflowCommand

Workflow guidelines for the action logic	• Extract parameters from the query string.
	For example:
	`rt.HttpRequest.QueryString["SelectedItems"]`
	• Design the forms to fit the dialog frame size.
	• Consider setting a meaningful *end message*.
	• Consider adding a timer to handle abandoned processes.
	• Set visibility to *hidden*.
		• Start with HTTP Input.
		• End with HTTP Output.
		• Extract parameters from the HTTP Input body.
		For example:
		`- Join({HTTP Input}.In["body"].SelectedItems, ",", true)`
		`- TryElse(ToString({HTTP Input}.In["body"].Parameters.where(Name=="myParam1").last().Value),"")`
		• Set visibility to *hidden*.
		• Consider setting the persistence mode to run in-memory.
		• The HTTP Listener *InvokeWorkflow* is a prerequisite for the backend custom actions.

Completion message	The workflow end message.	App notification pop up.

Configure custom actions

To configure custom actions, follow the steps below:

Step	Action
1.	Create the workflow with the logic of custom actions.
2.	Create a yaml (ConfigSet) with custom actions, for the grids where the custom actions should be available.
3.	Link between the ConfigSet and your Solution. • For HotOperations Solutions, use the solution wizard to set the ConfigSet. • For other implementations (like Cora Orchestration solution), use the `Portal.ConfigSetId` application variable to set the required ConfigSet.

Once all this is done, refresh the portal page to see the custom actions in the context menu or the toolbar in the portal.

1. Create and import the workflow and the ConfigSet

Use the custom action template *CoraOrchestrationTemplates* in this link to learn about the configuration options for custom actions. The package includes the following.

Type	Name	Comment
ConfigSet	CustomActionExample_ConfigSet.zip	Contains example of a yaml document for custom actions. Use this as a template for configuring custom actions. Contains examples of: • Frontend and backend actions. • Use of expressions. • Setting the order of the actions. • Removing standard actions. • Using display rules.
Workflow	CA Frontend Example1.zip	Sample workflow for a frontend custom action. Use this as a template for actions that require input screens from the user.
Workflow	CA Backend Example2.zip	Sample workflow for a backend custom action. Use this as a template for actions that do not require any input from the user. This workflow also imports the web service "**InvokeWorkflow**" that is mandatory for the backend actions.

2. Create yaml (ConfigSet) with custom actions

There are three main sections in the yaml document:

<table>
<tr><th>Section</th><th>Description</th></tr>
<tr><td>commandManagerId</td><td>Defines the relevant grid for the custom actions.
The available options are:
<table>
<tr><th>commandManagerId</th><th>Grid</th><th>Item Id</th></tr>
<tr><td>sq.ui.commands.ft.cases</td><td>My Operation Cases</td><td>$item.CaseId</td></tr>
<tr><td>sq.ui.commands.ft.teamLeaderTasks</td><td>My Team Cases</td><td>$item.CaseId</td></tr>
<tr><td>sq.ui.commands.ft.teamMemberTasks</td><td>Cases Assigned To Me and Cases In Queue</td><td>$item.CaseId</td></tr>
<tr><td>sq.ui.commands.ft.messages</td><td>My Tasks and My Notifications</td><td>$item.Id</td></tr>
<tr><td>sq.ui.commands.ft.instances</td><td>Process I Started and Process Page</td><td>$item.InstanceId</td></tr>
</table>
</td></tr>
</table>

Section	Description
commandTable	Defines the added custom actions. Types of custom actions: • OpenWorkflowWindowCommand - Frontend option for a custom action with a pop up form to get user input. Parameters are passed via a query string. • InvokeWorkflowCommand - Backend option for custom action. Parameters are passed via the web service "InvokeWorkflow". Configure the behavior of the actions: • Selection mode: ◦ Single: Only one item can be selected. ◦ Multiple: Only multiple items cab ne selected. ◦ Both: One or more items can be selected. • Selection limit: Number of maximum items that can be selected for batch actions. • Batch mode: Relevant for toolbar actions. ◦ True: Trigger the workflow once and pass it the list of selected items. ◦ False: Trigger the workflow for each selected item. • Title: Relevant for actions, set the title of the dialog frame. • Prompt: Set the prompt behavior for confirmation text and options. • Parameters: ◦ The first parameter must be the row item Id. Set value expression with the key name according to the grid you are working on (the parameter name property is optional). ◦ Add additional parameters as required by your workflow logic.

Section	Description
controlTable	Defines the location in the UI where the custom action is available, the **Context Menu** and the **Toolbar**. General structure: • ContextMenu ◦ ContextMenuItem X ◦ ContextMenuItem Y ◦ ContextMenuItem Z • ToolBar ◦ ToolBarMenuItemA ◦ ToolBarMenuItemB ◦ ToolBarMenuItemC ◦ Each menu item or toolbar item can include optional instructions for ordering the actions: ▪ Clear ▪ Remove ▪ Remove at ▪ Insert before ▪ Insert after ▪ Insert at ◦ Each menu item or toolbar item can include optional display rules to determine the conditions for having the action enabled.

Starting from V10.7, you can configure custom actions with display rules that determine the selected Process page process, stored view or view type:

The display rules can be applied on various context menu items and toolbar items.

Examples

Workflow space Id can be queried with the following:

```
rt.HttpRequest.Headers["WorkflowSpaceId"] == "GUID"
```

Solution Id can be queried with the following:

```
rt.HttpRequest.Headers["SolutionId"] == "GUID"
```

Selected stored view can be queried by name and type.

```
- type: rule
value: ':$context.StoredViewName=="caso"'
- type: rule
value: ':$context.StoredViewType=="shared"'
```

The view type can be queried with the following expression:

```
- type: Rule
value: ': rt.HttpRequest.Headers["viewtype"] == "active"'
- type: Rule
value: ': rt.HttpRequest.Headers["viewtype"] == "completed"'
```