# Module 4 Lab B

# Creating a Custom Workflow Action

##### Objective: use the Dynamics 365 VS Toolkit to create a custom workflow action that will check newly created accounts for duplicates and add the custom action to a workflow which will be tested and debugged

**Step 1: Creating the Visual Studio Solution Dynamics 365 project**

|  |  |
| --- | --- |
|  | Make sure that your Tools/Options D365 Tool paths are set to a V8 copy of the SDK (Out of the box PluginRegistration Tool and Bin directory) |
|  | Open Visual Studio 2017 Community Edition and create a new Dynamics 365 Visual Studio solution template.  Create a new solution. |
|  | Connect to your online CRM trial. |
|  | Create a new CRM Solution.  Select “Start from CRM” |
|  | Log in to your CRM solution |
|  | Select the default solution |
|  | Add new Templates |
|  | Add all three options. |
|  | Set the names |
|  | Check that new solution has been created:- |
|  | Go to the CRM Explorer, open to view the Account entity, right click and Choose “Add Workflow Activity” |
|  | Set the name to “Detect Duplicate Account” |
|  | Solution should appear as shown. |
|  | Edit “DetectDuplicate.cs”.  Remove the “Account” argument.  Add a PossibleDuplicate output  [Output("Bool output")]  public OutArgument<bool> PossibleDuplicate { get; set; } |

|  |  |
| --- | --- |
|  | Edit ExecuteCRMWorkFlowActivity method.  var tracingService = crmWorkflowContext.TracingService;  tracingService.Trace("Starting DetectDuplicate workflow activity");  if (crmWorkflowContext == null)  {  throw new ArgumentNullException("crmWorkflowContext");  }  try  {  tracingService.Trace($"Pre entity images {string.Join(",",  crmWorkflowContext.WorkflowExecutionContext.PreEntityImages.Keys)}");  var target = (Entity) crmWorkflowContext.WorkflowExecutionContext.PreEntityImages["account"];  tracingService.Trace($"Target attributes are:{string.Join(",",target.Attributes.Keys)}");  tracingService.Trace($"Target name is {target["name"]}");  var duplicates = NumberOfDuplicates(crmWorkflowContext.OrganizationService,  target.GetAttributeValue<string>("name"));  tracingService.Trace($"Number of duplicates {duplicates}");  if (duplicates > 0)  {  PossibleMatch.Set(executionContext, true);  }  else  {  PossibleMatch.Set(executionContext, false);  }  }  catch (FaultException<OrganizationServiceFault> e) |
| 1. **4** | Override the execute method by typing override within the DuplicateChecker Class and then selecting execute from the list displayed. |
| 1. **5** | We will replace the code that throws the exception with code that checks for duplicate accounts.  Before that we will need to provide the custom action with two parameters. An input parameter that will reference the account that has just been created and an output parameter that will flag the account as a possible duplicate.  Within the class but outside of the execute method declare two properties as listed below including the attributes that will mark the direction for these parameters.  [Input("EntityReference input")]  [ReferenceTarget("account")]  public InArgument<EntityReference> AccountReference { get; set; }  [Output("Bool output")]  public OutArgument<bool> PossibleDuplicate { get; set; } |
| 1. **6** | Your class should look like this |
| 1. **7** | Within the Execute method replace the statement that throws an exception with a variable declaration. Name the variable tracingService and set the type to ItracingService. Assign to it as follows  ITracingService tracingService = context.GetExtension<ITracingService>();  This will allow us to write diagnostic output to the trace listener.  Following that declare a variable named wfContext of type IWorkflowContext and assign to it as follows.  IWorkflowContext wfContext = context.GetExtension<IWorkflowContext>();  Through this object we will be able to access Workflow execution context information including the parameters that are used to pass information into and out of the custom action. |
| 1. **8** | Next declare a variable named serviceFactory of type IOrganizationServiceFactory. Then use it to get a reference to an object of type IOrganizatiionService which will allow us to perform CRUD operations on Dynamics 365. See below….  IOrganizationServiceFactory serviceFactory =  context.GetExtension<IOrganizationServiceFactory>();    IOrganizationService service  serviceFactory.CreateOrganizationService(wfContext.UserId);  The completed method should look like this. |
| 1. **9** | Below the code that you have just added but still within the Execute method declare an int named numberOfAccounts and initialise it to 0. This will be store the number of accounts with the same name in our system.  Following that declare a variable of type Entity named newAccount and assign it the account reference that was passed in as a parameter to the custom workflow activity. Limit the ColumnSet to just the name attribute.  Entity newAccount = service.Retrieve("account", this.AccountReference.Get(context).Id, new ColumnSet("name")); |
| 1. **10** | If newAccount is not null retrieve the account name from its attributes and store this value in a string variable named newAccountName.  You could also use the reference to the tracing service to log diagnostic info full code listing shown below shown below. |
| 1. **11** | Below the call to the tracingService’s trace method but still within the if statement use the OrganizationService’s retrieveMultiple method passing it an object of type FetchExpression or QueryExpression.  These should be configured search for all accounts that have the same name as the newly create account and store the number of matched accounts in the variable named numberOfAccounts you previously declared. Use the example below if you want, which uses the QueryExpression technique.  //build Query Expression to do search  ConditionExpression condition1 = new  ConditionExpression();  condition1.AttributeName = "name";  condition1.Operator = ConditionOperator.Equal;  condition1.Values.Add(newAccountName);  FilterExpression filter1 = new FilterExpression();  filter1.Conditions.Add(condition1);  QueryExpression query = new QueryExpression("account");  query.ColumnSet.AddColumns();  query.Criteria.AddFilter(filter1);  QueryExpression dupTest = new QueryExpression("account");  EntityCollection result = service.RetrieveMultiple(query);  //store the number of accounts with the same name  numberOfAccounts = result.Entities.Count;  //Send info to the trace log  tracingService.Trace("NumberOfDuplicates is " +  numberOfAccounts.ToString()); |
| 1. **12** | For the if statement create an **else** block that throws an exception with the message “Account parameter not set”. |
| 1. **13** | Below the current if else statement but still within the Execute method create another if else block based on the value of the numberOfAccounts variable.    If the numberOfAccounts is greater than 1 then we have a possible duplicate account created so set the boolean output parameter to true else set it to false. |
| 1. **14** |  |

**Step 2: Registering the custom Workflow Activity (Custom Action)**

|  |  |
| --- | --- |
| **1** | Build you project and fix any errors. Next go to the project properties and create a new key to sign your completed assembly. Give the snk file any name you like but don’t bother protecting it with a password. |
| **2** | Rebuild the project again making sure that you have selected the Debug configuration. |
| **3** | Now **copy** the .snk file previously created (you’ll find it in the project folder) to the Dynamics 365 sdk \tools\PluginRegistration folder.  Rename the copy PluginProfiler.DefaultWorkflowAssembly.snk. This will allow us to debug the assembly later.    The file should now be located here  C:\Program Files\Microsoft SDKs\SDK\Tools\PluginRegistration\ PluginProfiler.DefaultWorkflowAssembly.snk |
| **4** | Run the Plugin registration tool PluginRegistration.exe and click on Create New Connection. You will find this in the SDK\Tools folder.  Connect to your organisation    Click on show advanced and select your region entering your username and password, ensuring that you have selected the office365 option at the top for deployment type    If prompted to save the connection as a named connection, feel free to enter an appropriate value |
| **5** | Next click on the install profiler button at the top    This may take a while but we will need the profiler to debug our custom workflow activities and plugins.  When the install is complete you will see it in the list of plugins and workflow activities. |
| **6** | Now we will use the tool to register our assembly and plugin. Click on the register button at the top, then click Register New Assembly. |
| **7** | Navigate using the browse button to your bin/debug folder and select the CustomWFAction.dll file, ensure the other options are as shown in the diagram below.    When completed, you should see a dialog similar to that shown below    Now that the Custom Workflow activity is registered we can use it within a workflow process. |

**Step 3: Creating a workflow that uses the custom activity**

|  |  |
| --- | --- |
| 1 | Navigate to the Settings area of your Dynamics 365 org and under the Customization category select the Solutions tile.    Create a new solution.  Name the Solution Module4, set the version number to 1.0.0.0 and select the publisher using the browse button |
| 2 | Click on the process node on the left-hand side and the click new |
| 3 | Choose to create a workflow, named “Check for Duplicate Accounts”, associated with the Account entity and select New blank process option |
| 4 | In the next dialog leave the default options selected as seen below |
| 5 | Scroll down the current window until you see a drop down labelled Add Step and from that drop down click on CustomWFAction(1.0.0.0) > CustomWFAction.DuplicateChecker    For the description enter “Checking for duplicate accounts”. |
| 5 | Now we will bind the input parameter to the Account entity that triggered the Workflow. Click on the Set Properties button. |

|  |  |
| --- | --- |
|  | Using the Form Assistant on the right make sure Account is selected (Both drop downs) and then click Add.    Select a default value of A Datum then click on OK. (The actual value will set when the workflow executes with the context of a newly created account) |
| 6 | Click on the save and close button. Continue to edit the Workflow by adding a check condition as shown below.      For the description type “Do we have duplicates ?”. Then click on the <condition> (click to configure) link |
| 7 | Click on the Select link    Scroll to the bottom of the window that has opened and under local variables select Checking for duplicate Accounts. |

|  |  |
| --- | --- |
| 8 | In the Select link to the right, select the item labelled bool output then to the right of that another select should be set to Equals    Click on the Enter Value link at the end of the line and select True. The click on save and close.  This step has called the custom workflow action to check for duplicates. |
| 9 | Click on the select this Row and click Add Step |
| 10 | From the options available select Create Record    In the description type “Create a task to prompt user to check if this is a duplicate account”.  From the Create drop down select task and then click on set properties.    In the Subject field enter “This may be a duplicate account” notice the regarding field is filled in appropriately. |
| 11 | Click on save and close then Activate the Workflow by clicking on the activate button.    Close the window then. Within the Solution select to publish all customizations. |
| 12 | Now test the work flow by creating an account named AAA. You should check that there are no tasks that have been created by the workflow to check for duplicates |
| 13 | Create a second account named AAA. This one should have a task that prompts the user to check for duplicates. If your workflow doesn’t appear to function correctly, then the next section on debugging should resolve any issues (hopefully!) |

**Step 4: Debugging the Custom Workflow Activity (Action)**

|  |  |
| --- | --- |
| 1 | We will now set up a debug session to step through the code generated for the workflow activity that you deployed in the previous exercises.  Open the Plugin Registration tool and right click on the Plugin Profiler node and select “Start Profiling Workflow”. |
| 2 | Enter values into the dialog as shown below and then click on OK. |
| 3 | Below the Plug-in Profiler node, you should now see that the Check for duplicate accounts workflow activity is now configured for profiling. |
| 4 | Now use the dynamics portal to create another duplicate account |
| 5 | Navigate to the settings area a select and then click on the Plug-in Profiles button under the Extensions category. |
| 6 | Click on the record that has created |
| 7 | Expand the field named Serialized Content and carefully copy **all** of the contents to a notepad file.    Save it to a folder somewhere you can easily get to e.g. Create a folder named debug on your c: drive call the file debugWorkflow.txt or something similar. |
| 8 | Back in the plugin Registration tool click on the debug button. |
| 9 | Load the file that you created earlier in step 8. Also select the assembly located in the bin debug folder of the visual studio project.      **Leave this dialog open**. |
| 10 | Place a breakpoint somewhere in the execute method as shown below |
| 11 | On the debug menu in Visual Studio select to attach to the Plugin Registration Tool |
| 12 | Go back to the Plugin Registration Tool and the Replay Plugin window that you had left open and click on the start execution button. |
|  | You should now be able to replay the code that executed for that Account creation operation. The debug tools that allow you to step through your code and investigate the value of variable in scope are all available. |
|  |  |