# 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** | Add a separate NumberOfDuplicates method. We will make this public so that we can unit test at a later date.  public int NumberOfDuplicates(IOrganizationService service, string name)  {  var qe = new QueryExpression("account")  {  Criteria = new FilterExpression  {  Conditions =  {  new ConditionExpression  {  AttributeName = "name",  Operator = ConditionOperator.Equal,  Values = {name}  }  },  },  ColumnSet = new ColumnSet(),  };  var result = service.RetrieveMultiple(qe);  return result.Entities.Count;  } |
| 1. **5** | Update Nuget packages to 9.0.2.5 |
| 1. **SW** | Sign the assembly |
| 1. **6** | Rebuild the solution. Then we are ready to deploy. |

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

|  |  |
| --- | --- |
| **1** | Right click on CRM Package and select “Deploy” |
| **2** | Select just the DetectDuplicate workflow to deploy. |
| **3** | Check that deployment succeeds |
| **4** | Open plugin registration tool, and check that the deployment has worked. |

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

|  |  |
| --- | --- |
|  | 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 |
|  | Click on the process node on the left-hand side and the click new | |
|  | Choose to create a workflow, named “Check for Duplicate Accounts”, associated with the Account entity and select New blank process option | |
|  | In the next dialog leave the default options selected as seen below | |
|  | Scroll down the current window until you see a drop down labelled Add Step and from that drop down click on CRMTraining and select the “DetectDuplicate” custom activity.    For the description enter “Checking for duplicate accounts”. | |
|  | Click on the Set Properties button. | |
|  | Note that there are no properties to set – the identity of the account is taken from the workflow context. | |

|  |  |
| --- | --- |
| 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. |
|  |  |