Table of Contents

List of Figures 1
Introduction 2
Initial setup 2
Custom Activity Arguments 2
1. Invoke Code Activity 2
2. Create a reusable library of activities 3
   A. Local repository preparation 3
   B. Creating activities in the library 4
   C. Publishing the library for further reuse 5
   D. Installing and using an activity library package in UiPath Studio 5
3. Creating a C# class library of activities 6
   A. Creating activities in a C# class library 6
   B. Creating the packages in NuGet Package Explorer 8
   C. Package installing in UiPath 9

List of Figures

Figure 1. Invoke Code Activity – sections 2
Figure 2. Invoke Code activity – setting arguments 3
Figure 3. Invoke Code activity – editing code 3
Figure 4. Register a new folder of *User defined package sources* 4
Figure 5. InsertToList activity definition in the Library 4
Figure 6. GetItem activity definition in the Library 4
Figure 7. Library publishing options window 5
Figure 8. Installing a library package 5
Figure 9. Defined activities usage 6
Figure 10. Class Library project creation in Visual Studio 6
Figure 11.Adding references to the Class Library project 7
Figure 12. Adding class activities to the class library 7
Figure 13. Package configuration in NuGet Package Explorer 9
Figure 14. Save as... window for the package 9
Figure 15. Installing the class library in UiPath Studio 10
Figure 16. Class library available to use in UiPath Studio 10

Introduction

This tutorial presents three ways to customize activities and use them in UiPath workflows:

1. make use of **Invoke Code** activity ;
2. create a **library** in UiPath, publish and import it where required;
3. create a **class library** in C#, generate the associated NuGet package and import it where required.

Initial setup

As workflow to work on we will consider **Demo11-ListCustomActivities**. The workflow emphasizes List operations using activities. The predefined package of activities that works on collections include activities for the following operations: Add, Remove, Exists, Clear. The package does not include activities for the Insert(index, Object) and Get(index) operations. In **Lecture 03** the **Invoke Method** activity was used to call the Insert method. Still, these methods are good candidates to show how custom activities can be created.

Custom Activity Arguments

The Insert method allows inserting an item on a specific index into an existing collection. Therefore, the activity will have the following arguments:

- [in/out] `drinksList: List<Object>;`
- [in] `index: Int32;`
- [in] `currentDrink: Object;`

The Get method allows accessing the item available on a given index from an existing collection. Therefore, the activity will have the following arguments:

- [in] `drinksList: List<Object>;`
- [in] `index: Int32;`
- [out] `item: Object;`

1. Invoke Code Activity

This activity allows adding custom code written in **VB.Net**. Steps required:

1. Search and add to the workflow the **Invoke Code** activity; it should allow to adapt two sections:
   a. the input/output arguments (see Figure 1);
   b. the source code (see Figure 1).

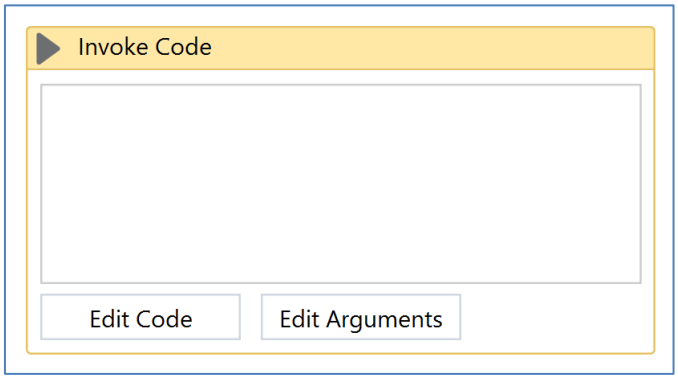

Figure 1. Invoke Code Activity – sections

2. Edit the *arguments* for the code that will be added; this is similar to editing arguments for workflows (task required for **Lab02**) (see Figure 2);

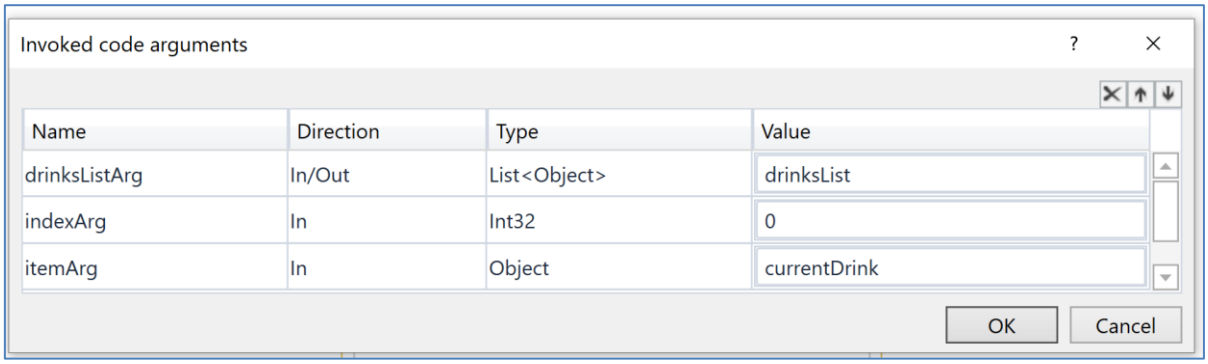

Figure 2. Invoke Code activity – setting arguments

3. Add the following code that performs the action:

```
drinksListArg.Insert(indexArg, itemArg)
Console.WriteLine("[Invoke Code activity]:"+" item "+itemArg.ToString+" was inserted on index "+indexArg.ToString)
```

This should look like the one in Figure 3.

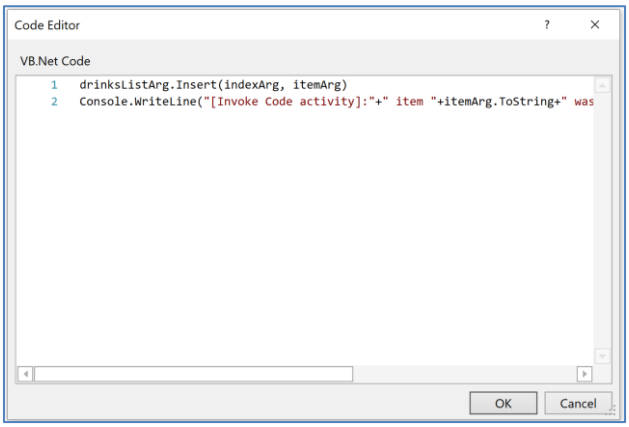

Figure 3. Invoke Code activity – editing code

4. Run the code as usual; make sure the initial *insert* operation included in **Invoke Method** activity is commented (Ctrl+D).

2. Create a reusable library of activities

This approach allows creating a library of multiple custom activities that will be published in a local folder and used when required, similar to existing activities. Steps required:

A. Local repository preparation

1. Create a folder that will indicate a local repository of the published activities. E.g., c:\CustomActivities.
2. Register the folder as follow:
   - in the **Design** panel, open **Manage Packages** option;
   - in the **Settings** option fill in the details of te repository:
     - **Name: CustomActivities**
     - **Source: c:\CustomActivities**
   - click **Add**;
   - the folder will be added in the *User defined package sources* list; it will be added in the short list available on the left hand side of the **Manage Packages** feature (see Figure 4);

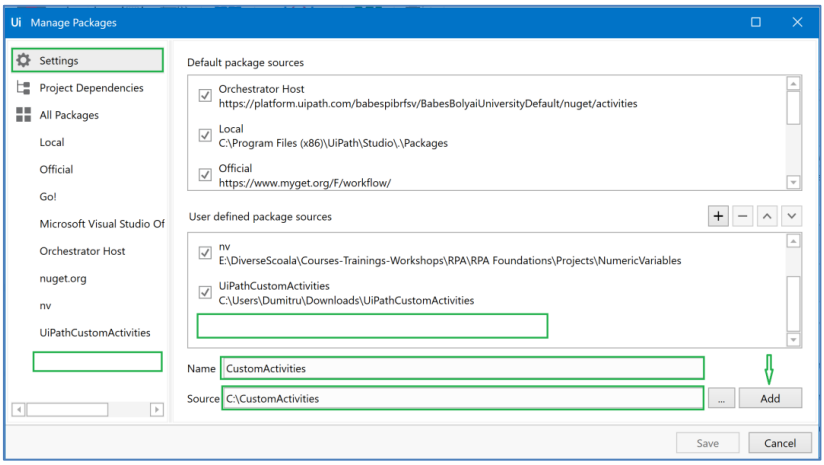

Figure 4. Register a new folder of *User defined package sources*

B. Creating activities in the library

1. Go the **Start** option and create a **Library project**, e.g., ListOpsLibrary;
   - the steps are similar to creating a **Process** in UiPath Studio;
   - the difference between this two types of projects are:
     - **Library:** the workflow(s) defined will be used similar to an already defined activity; this is the main purpose of using them;
     - **Process:** the workflow(s) defined can be executed immediately, without additional preparation;
     - **Library:** the workflow(s) cannot be run by it/themselves; an exception is thrown;
     - **Process:** the workflow(s) can be run by it/themselves or by using **Invoke Workflow File** activity;
2. Add an **.xaml** file for each activity that it is intended to be included in the library; e.g., the activities GetItem and InsertToList will be created as configured as in Figure 5 and Figure 6.

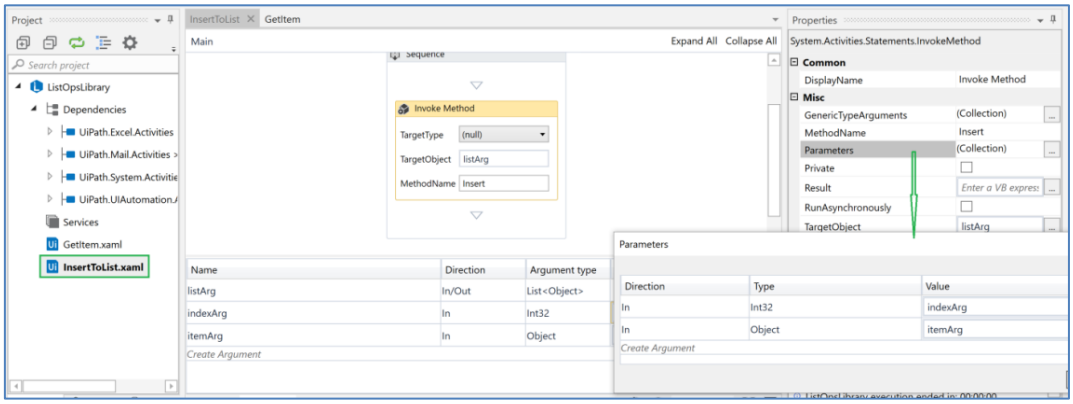

Figure 5. InsertToList activity definition in the Library

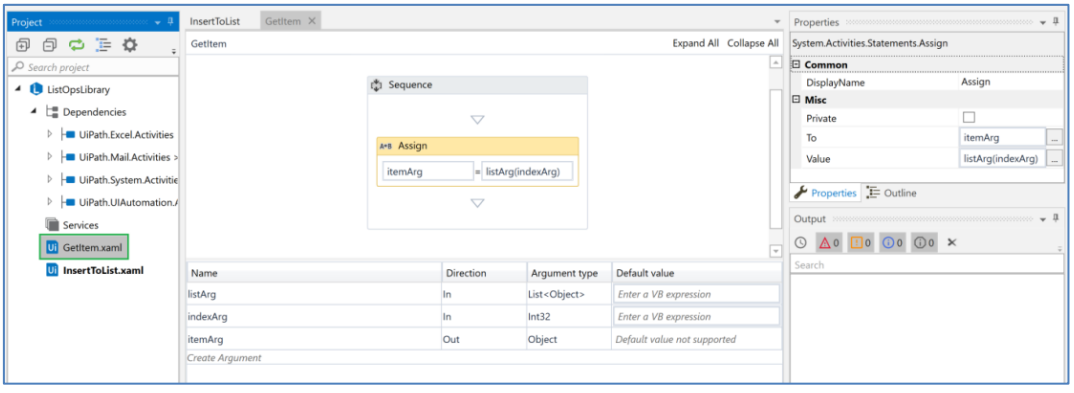

Figure 6. GetItem activity definition in the Library

C. Publishing the library for further reuse

1. In the **Design** panel, click **Publish** option:
   - **Publish Library** allows to customize the library to be published; filling in:
     - **Custom URL:** the folder that will contain the **.nupkg** file generated when publishing a library, e.g., **c:\CustomActivities**;
     - other information like version, release notes, etc. (see Figure 7);
   - click **Publish**;
2. In order to apply changes to a published library, this should be re-published, by following the previous step; a new version for the same library is generated;

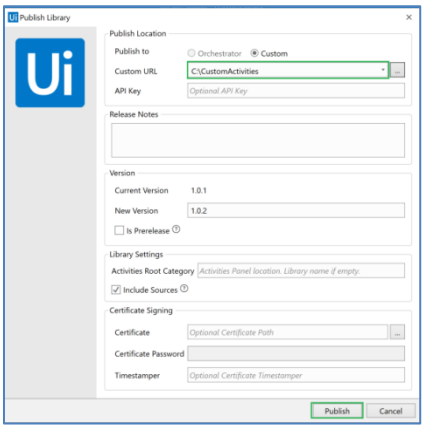

Figure 7. Library publishing options window

D. Installing and using an activity library package in UiPath Studio

1. This action is performed in a similar way as for predefined activities;
   - the folder **c:\CustomActivities** contains the package of the library published before, e.g., ListOpsLibrary.1.0.5.nupkg;
2. In the **Design** panel, open **Manage Packages** option;
3. In the left hand side list of activity repositories, choose **CustomActivities**;
   - the list of available packages is updated with the distinct packages; if updates are available for the installed ones, they are highlighted;
4. Select the appropriate library package;
5. Click **Install** and **Save** (see Figure 8);
   - in case of available upgrades for an already installed package, click **Update** and **Save**;
6. The library should be available for use as in Figure 9;

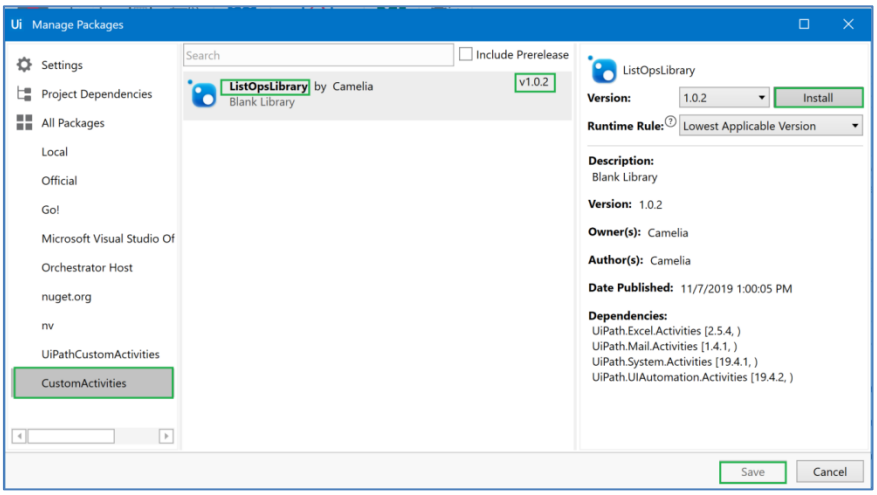

Figure 8. Installing a library package

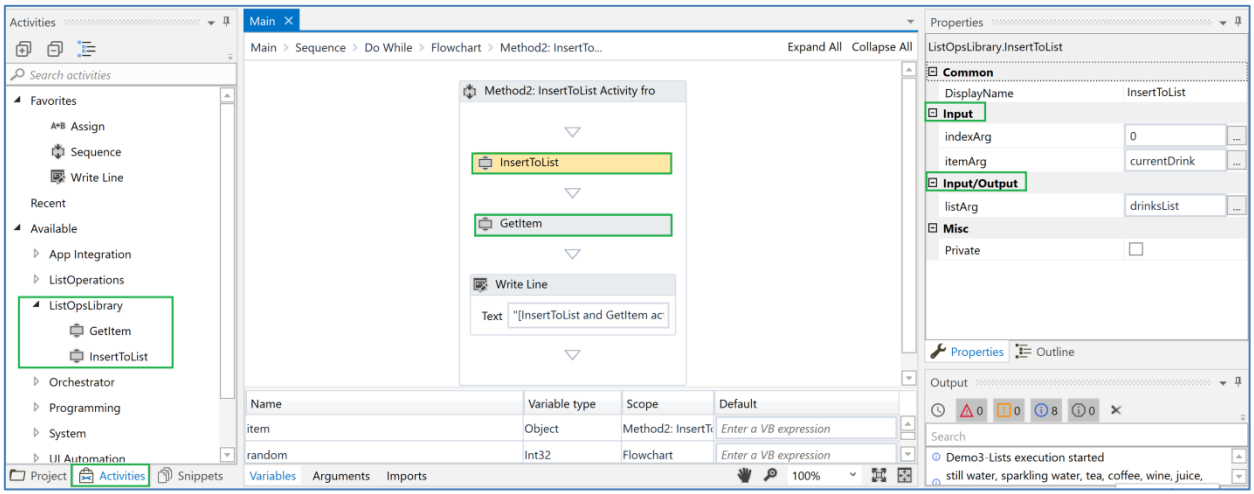

Figure 9. Defined activities usage

3. Creating a C# class library of activities

This approach allows creating a package of custom activities that will be published as a package in a local folder and used when needed. C# programming language is used to create the class library.

A. Creating activities in a C# class library

1. Create a project in Visual Studio, by filling in:
   - **Language: Visual C#**;
   - **Type of the project: Class Library**
   - **Name: ListActivities**;
   - **Location: C:\CustomActivities** (see Figure 10);
2. Click **OK**;

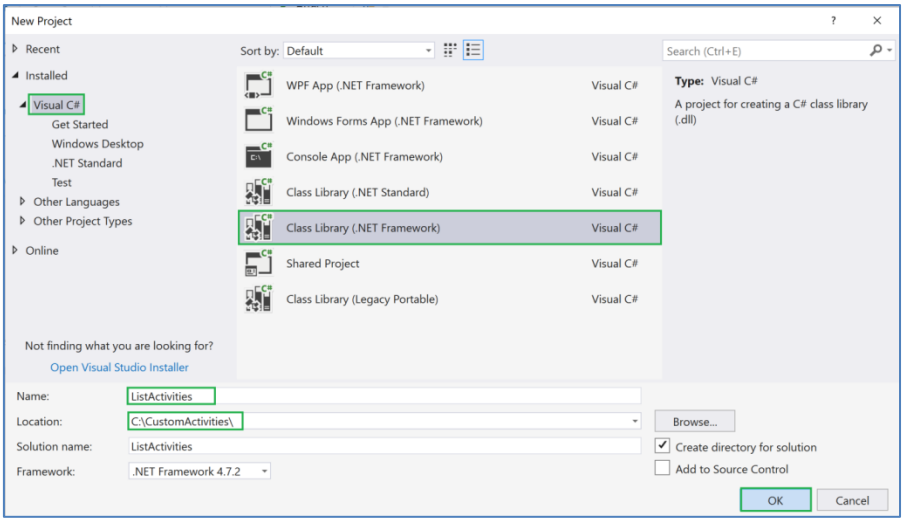

Figure 10. Class Library project creation in Visual Studio

3. Add the dependencies to the created project:
   - right-click on the project name, in the **Solution Explorer**;
   - choose **Add**, then choose **Reference**...;
   - select the dependencies:
     - System.Activities;
     - System.ComponentModel.Composition (see Figure 11);
   - click **OK**;

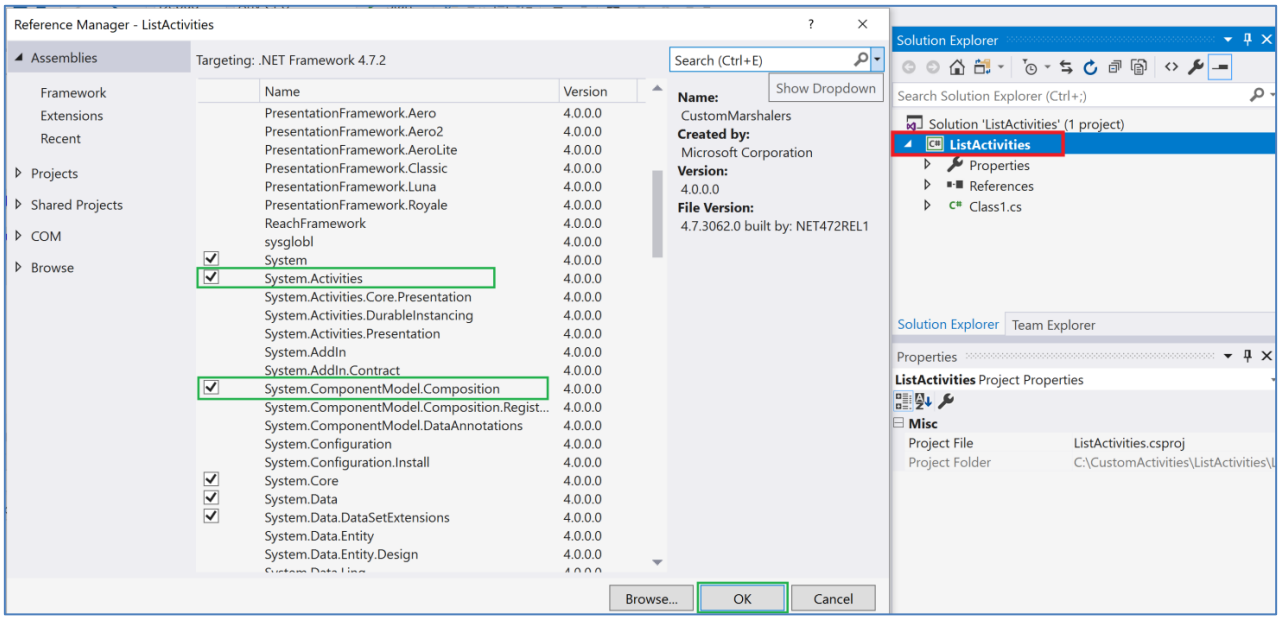

Figure 11.Adding references to the Class Library project

4. Add classes for activities:
   - indicate the use of Activities and ComponentModel in the source code;
   - the name of the namespace is the name of the future activity library, e.g., ListActivities;
   - for each desired activity, **a new class is added** in the namespace; each class is an activity, therefore it inherits from CodeActivity class and overrides its Execute method (see Figure 12);

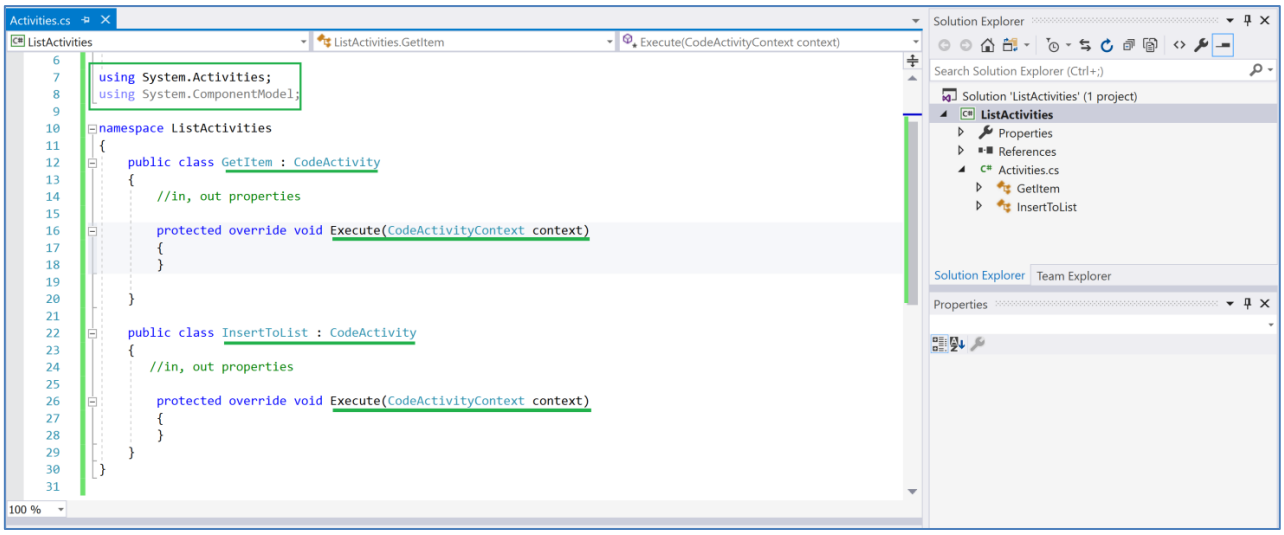

Figure 12. Adding class activities to the class library

5. Perform changes to the activities:
   - for **Get Item** activity the class should be as follows:

```
public class GetItem : CodeActivity
    {
        [Category("Input")]
        public InArgument<List<Object>> Container { get; set; }

        [Category("Input")]
        public InArgument<int> Index { get; set; }

        [Category("Output")]
        public OutArgument<Object> Item { get; set; }

        protected override void Execute(CodeActivityContext context)
        {
            List<Object> l = Container.Get(context);
            int index = Index.Get(context);
            Item.Set(context, l.ElementAt(index));
        }
    }
```

- For **Insert To List** activity the class should be as follows:

```
public class InsertToList : CodeActivity
    {
        [Category("Input/Output")]
        public InOutArgument<List<Object>> Container { get; set; }

        [Category("Input")]
        public InArgument<int> Index { get; set; }

        [Category("Input")]
        public InArgument<Object> Item { get; set; }

        protected override void Execute(CodeActivityContext context)
        {
            List<Object> l = Container.Get(context);
            int index = Index.Get(context);
            Object item = Item.Get(context);
            l.Insert(index, item);
        }
    }
```

6. Change the configuration from **Debug** to **Release** and build the solution (menu **Build**, **Build Solution**);
    - the file **ListActivities.dll** is created in the class library project in the folder **Release**;
    - this file will be used to create a **.nupkg** file in the following step;

B. Creating the packages in NuGet Package Explorer

1. download and install **NuGet Package Explorer**;
2. open the application and choose option **Create New Package**;
3. right-click and choose from the pop-up menu and choose **Add Extisting File…** option to add the **.dll** file located in the **Release** folder of the class library project, e.g., c:\CustomActivities\ListActivities\ListActivities\bin\Listactivities.dll;
4. edit the metadata from **Edit menu**, **Edit Metadata** option;
    - **Id: provide a name for the package, with the suffix Activity**; this allows UiPath Studio to recognize the package as containing activities, e.g., **ListActivities.Activity**;
    - other attributes of the package may be edited; some of them will be available for the user in UiPath Studio, e.g., version, summary, description, etc., (see Figure 13);
    - save the changes using the **green save** sign on the top of the editing area;

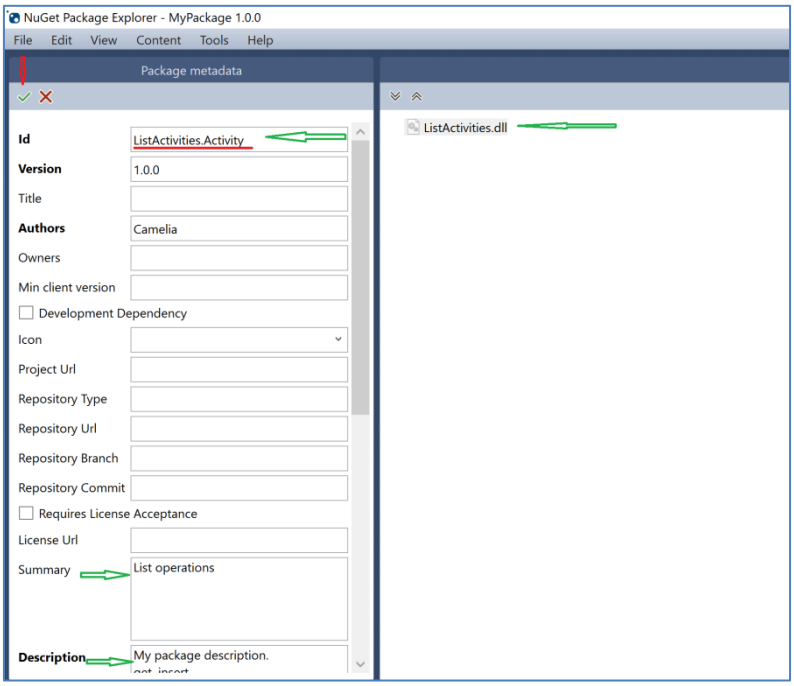

Figure 13. Package configuration in NuGet Package Explorer

5. Save the package:
   - the name of the package reflects the **id** and the **version** of the package;
   - the folder should the one indicated in UiPath Studio as the folder that consists of the *User defined package sources*, i.e., **C:\CustomActivities**;
   - the **Save as…** window should look like in Figure 14;
   - click **Save**;
   - after any change in the class library project a new version of the package should be built;
   - the package should be created and made available to use in UiPath Studio;

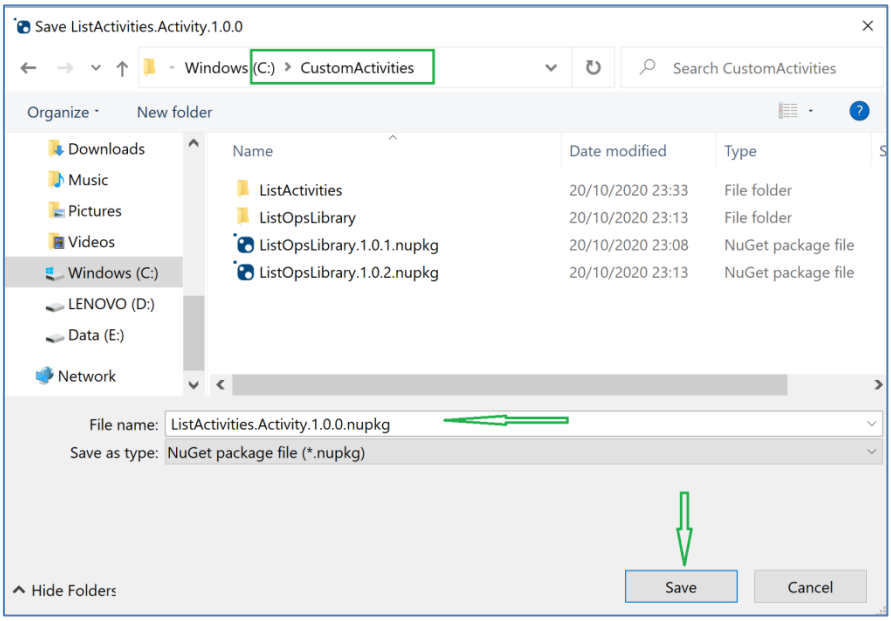

Figure 14. Save as... window for the package

C. Package installing in UiPath

1. in the **Design** panel, open **Manage Packages** option;
   - in the left hand side list of activity repositories, choose **CustomActivities**;
   - the list of available packages is updated with the distinct packages;

- if updates are available for the installed ones, they are highlighted;
- select the appropriate library package, e.g., **ListActivities.Activity**;
- click **Install** and **Save** (see Figure 15);

2. in case of available updates for an already installed package, click **Update** and **Save** (see Figure 16);

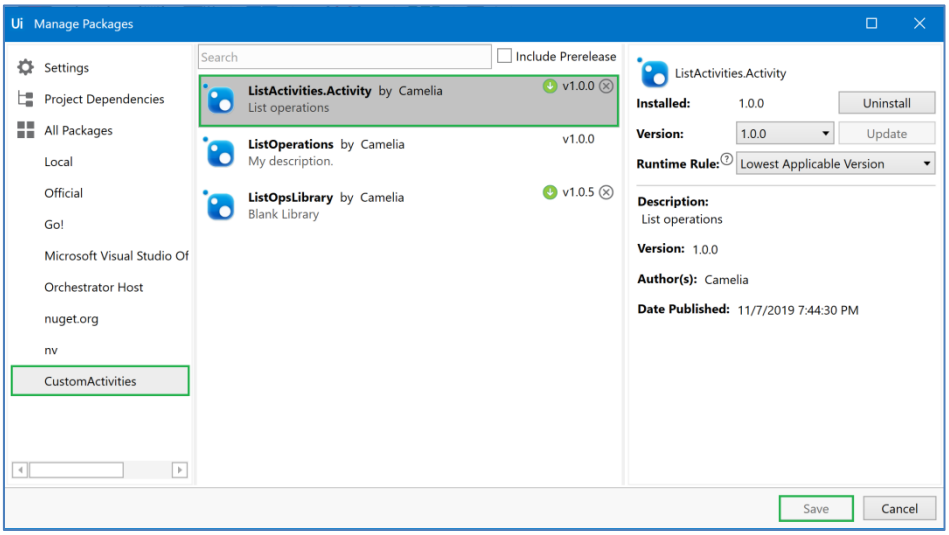

Figure 15. Installing the class library in UiPath Studio

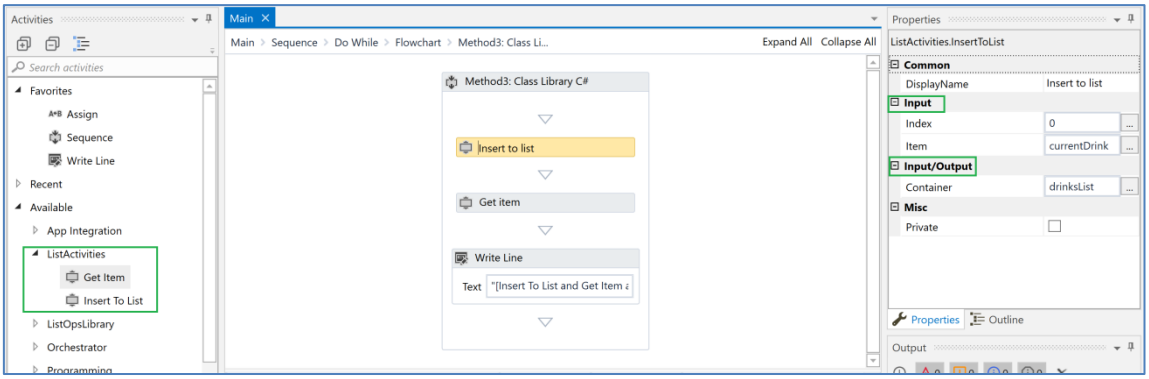

Figure 16. Class library available to use in UiPath Studio