

Proiect cofinanțat din FSE prin Program Operațional Capital Uman 2014 - 2020

Axa Prioritară 6 - Educație și competențe

Ațiune: OS6.7, OS6.9, OS6.10

Titlu proiect: *Antreprenor UBB!*

Acronim proiect: AUBB

Cod proiect: POCU/379/6/21/124662

Beneficiar: Universitatea Babeș-Bolyai din Cluj-Napoca

Activități definite de utilizator

O activitate proprie sau personalizată este o activitate care este definită de utilizator (programator RPA) și prin care se identifică acțiuni ce pot fi abstractizate și aplicate într-un context general, sub forma unor activități.

Acest material prezintă două metode de definire a activităților proprii (custom activities) care pot fi utilizate în procesele implementate în UiPath Studio. Cele două modalități prezentate sunt:

- activitatea **InvokeCode**;
- pachete de activități definite în UiPath Studio folosind un proiect de tip *Library*;
- pachete de activități definite în C# folosind biblioteci dinamice (dynamic-link libraries, dll).

Contextul general de definire a activităților personalizate

Pentru demonstrarea definirii activităților personalizate vom considera tipul de date **List**. În UiPath Studio este disponibil un pachet de activități necesare pentru descrierea operațiilor asociate unei liste, cum ar fi: **Add**, **Remove**, **Exists**, **Clear**. Pachetul de activități nu conține și activitățile asociate *inserării* sau *accesării* unui element în listă. Acest inconvenient poate fi depășit prin utilizarea activității **InvokeMethod** care va apela metoda explicită **Insert** și respectiv **Get**. Totuși, cele două operații sunt potrivite pentru a exemplifica modalitățile de implementare a activităților definite de utilizator.

Specificațiile asociate celor două operații sunt:

- metoda **Insert** permite introducerea pe o anumită poziție a unui element indicat:
 - `Insert ([in/out] list: List<Object>, [in] index: Int32, [in] item: Object);`
- metoda **Get** permite returnarea valorii unui element de pe o anumită poziție:
 - `Get ([in] list: List<Object>, [in] index: Int32, [out] item: Object);`

Utilizarea activităților personalizate definite de utilizator

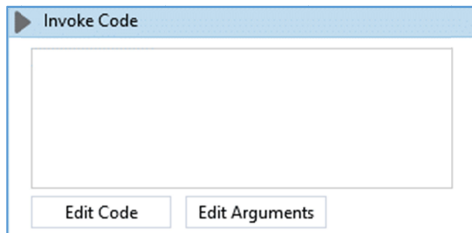
Problema Chelnerului amabil. Se consideră un proces care gestionează o listă de băuturi (`List<String>`) pe care un chelner le poate oferi clienților. Se generează aleator o valoare ce reprezintă poziția în lista a unei băuturi. Chelnerul oferă băutura clientului. Clientul o poate accepta sau refuza, iar chelnerul îi oferă o altă băutură, identificată aleator. O băutură poate fi oferită doar o singură dată. Băuturile acceptate și refuzate sunt memorate în liste separate. Dacă clientul refuză două băuturi atunci chelnerul îi spune “Multumim pentru vizită!” și îi prezintă lista de băuturi refuzate. Dacă clientul acceptă două băuturi chelnerul îi spune “Vă mai așteptăm!” și îi prezintă lista de băuturi permise.

I. Utilizarea activității `InvokeCode`

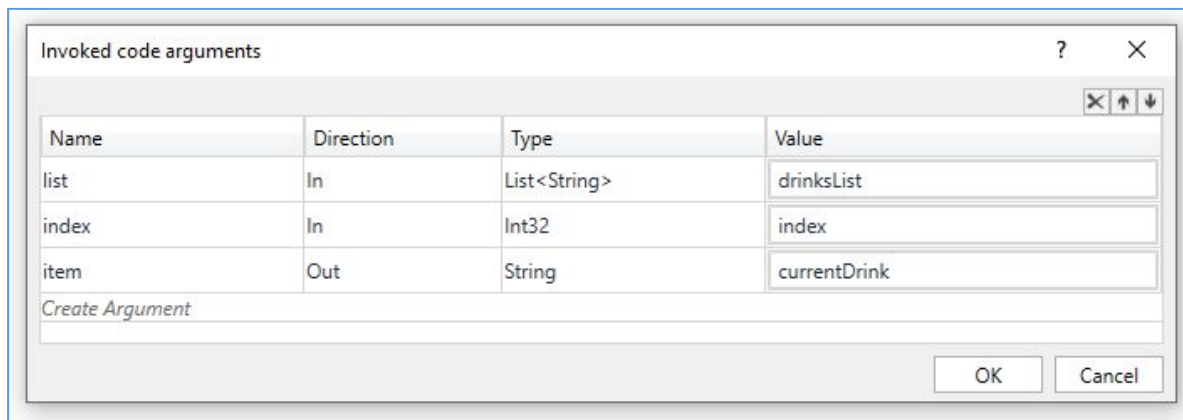
Activitatea `InvokeCode` permite adăugarea de cod sursă adaptat/personalizat, scris în limbajul VB.Net. Pași de utilizare:

Pas01. Se inserează o activitate `InvokeCode`. În cadrul activității se pot realiza două acțiuni (vezi figura de mai jos):

- **Edit Arguments** – pentru stabilirea parametrilor de intrare/ieșire;
- **Edit Code** – pentru editarea codului sursă.



Pas02[Get]. Pentru operația **Get** setarea parametrilor se realizează furnizând valori concrete pentru parametrii: *lista*, *poziția* de accesare și variabila care va stoca *elementul* returnat, ca în imaginea de mai jos.



Pas03[Get]. Pentru operația **Get** editarea codului sursă se realizează apelând metoda de accesare a unei valori pentru un obiect de tip **List**, folosind sintaxa VB.Net, ca în imaginea mai jos.

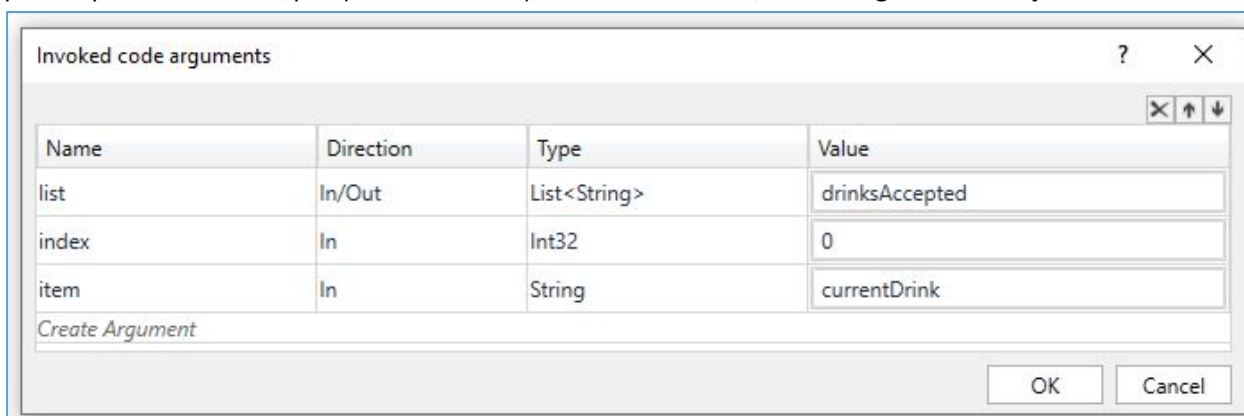
`item=list.Item(index)`

```
Console.WriteLine("item retrieved from "+index.ToString+" is "+item)
```



```
Code Editor
VB.Net Code
1 item=list.Item(index)
2 Console.WriteLine("item retrieved from "+index.ToString+" is "+item)
```

Pas02[Insert]. Pentru operația **Insert** setarea parametrilor se realizează furnizând valori concrete pentru parametrii: *lista*, *poziția* de inserare și *elementul* inserat, ca în imaginea de mai jos.



Name	Direction	Type	Value
list	In/Out	List<String>	drinksAccepted
index	In	Int32	0
item	In	String	currentDrink

Create Argument

OK Cancel

Pas03[Insert]. Pentru operația **Insert** editarea codului sursă se realizează apelând metoda de inserare a unei valori pentru o poziție indicată, utilizând sintaxa VB.Net, ca în imaginea mai jos.

```
list.Insert(index, item)
```

```
Console.WriteLine("item added at index "+index.ToString +", new size "+list.Count.ToString)
```



```
Code Editor
VB.Net Code
1 list.Insert(index, item)
2 Console.WriteLine("item added at index "+index.ToString +", new size "+list.Count.ToString)
```

II. Utilizarea pachetelor de activități definite în UiPath

Această abordare permite dezvoltarea de pachete de activități personalizate (custom activity libraries) care să fie publicate într-un folder local și folosite la nevoie. Utilizarea acestora se realizează în aceeași manieră în care se folosesc activitățile uzuale, disponibile la cerere în UiPath Studio.

Pentru a dezvolta pachete de activități în UiPath Studio trebuie să parcurgem următoarele etape:

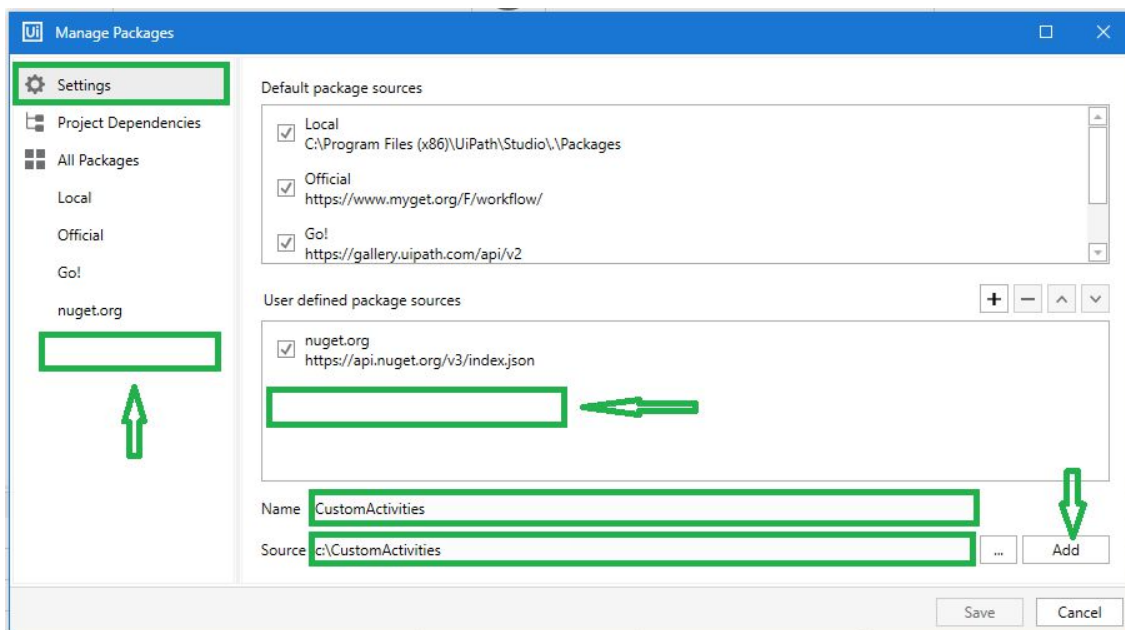
1. Pregătirea folderului în care se vor stoca pachetul de activități (local activity library repository);
2. Crearea pachetului de activități personalizate;
3. Publicarea pachetului de activități personalizate;
4. Instalarea și utilizarea pachetului de activități personalizate.

1. Pregătirea folderului care stochează pachetul de activități personalizate

Pas01. Se creează un folder care va reprezenta directorul în care se vor stoca pachetele de activități personalizate publicate, de exemplu, *c:\CustomActivities*.

Pas02. Se înregistrează acest director în UiPath Studio ca fiind folderul de depozitare a activităților personalizate.

- din meniul **Design** se alege opțiunea **Manage Packages**;
- în opțiunea **Settings** se completează informațiile referitoare la folderul care se înregistrează:
 - **Name:** *CustomActivities*
 - **Source:** *c:\CustomActivities*
- se finalizează adăugarea prin **Add**;
- folderul se adaugă implicit în lista *User defined package sources* și în lista folderelor cu accesare rapidă, disponibilă din partea stângă a ferestrei **Manage Packages**, ca în figura de mai jos.



2. Crearea pachetului de activități personalizate

Pas01. Din bara de meniuri se alege **Home** și opțiunea **Start**. Se alege apoi tipul de proiect *Library*. Se completează *numele proiectului*, *folderul* în care va fi memorat și *descrierea* activităților care vor fi incluse în cadrul proiectului de tip bibliotecă de activități.

Diferența dintre tipul de proiect *Library* și *Process* este dată de următoarele aspecte:

- obiective de utilizare:
 - *Process*: permit automatizarea și rularea imediată a unui proces, fără pregătiri suplimentare.
 - *Library*: permit descrierea de activități definite de utilizator care sunt integrate (apelate) în proiecte de tip *Process*.
- execuția proceselor (workflows):
 - *Process*: se pot executa din cadrul proiectului sau folosind activitatea **InvokeWorkflowFile**.
 - *Library*: nu se pot executa din cadrul proiectului; se va arunca o excepție la rulare.

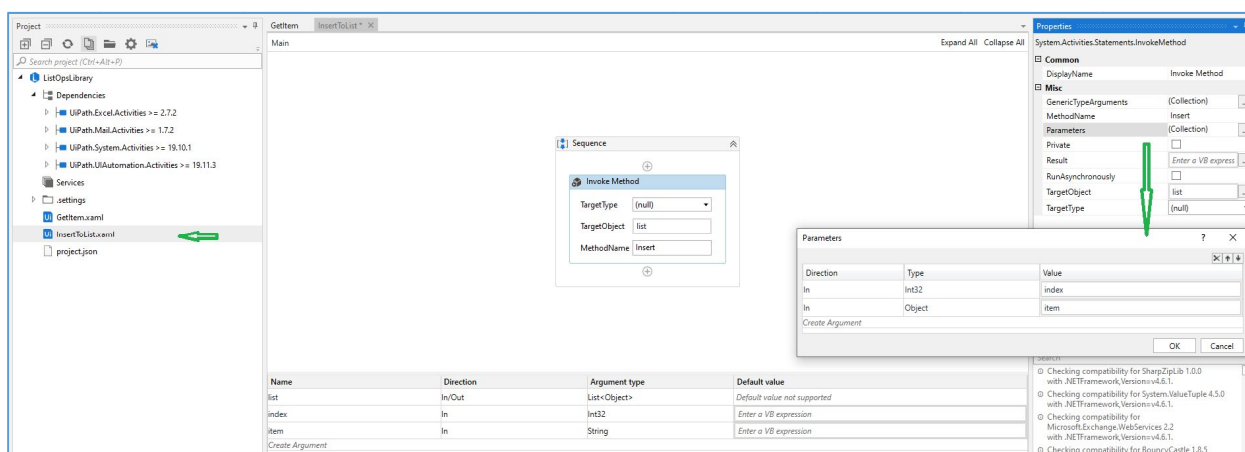
Pas02. În proiectul creat se adaugă câte un fișier **.xaml** pentru fiecare activitate care urmează a fi definită.

Pas02[Insert]. Pentru definirea activității **InsertToList** se adaugă o activitate **InvokeMethod** care va fi configurată astfel:

- În *Arguments Panel* se definesc trei parametri:
 - [in/out] list: `List<Object>`;
 - [in] index: `Int32`;
 - [in] item: `Object`;

- parametrii definiți în *Arguments Panel* vor reprezenta atribute ale activității **InsertToList**.
- În *Properties Panel* se configurează atributele:
 - **Collection**, indicând cei doi parametri ai metodei **Insert** (*poziția și elementul* adăugat);
 - **Target Object** = list
 - **Method** = Insert.

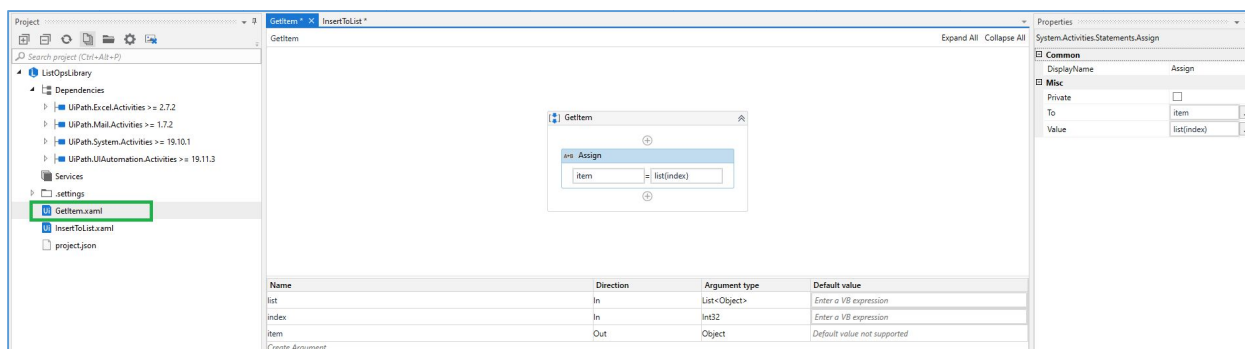
Pentru definirea activității **InsertToList** nu este necesară definirea de variabile, doar utilizarea parametrilor *in/out* și *in* asociați activității definite, ca în figura de mai jos. Totuși, în cazul în care definirea activității presupune utilizarea de date stocate în variabile locale, acestea se definesc în *Variables Panel*.



Pas02[Get]. Pentru definirea activității **GetItem** se adaugă o activitate **Assign** care va fi configurată astfel:

- În *Arguments Panel* se definesc trei parametri:
 - [in] list: List<Object>;
 - [in] index: Int32;
 - [out] item: Object;
 - parametrii definiți în *Arguments Panel* vor reprezenta atribute ale activității **GetItem**.
- Activitatea **Assign** se definește astfel:
 - item = list (index)

Pentru definirea activității **GetItem** nu este necesară definirea de variabile, doar utilizarea parametrilor *in/out* și *in* asociați activității definite, ca în figura de mai jos.

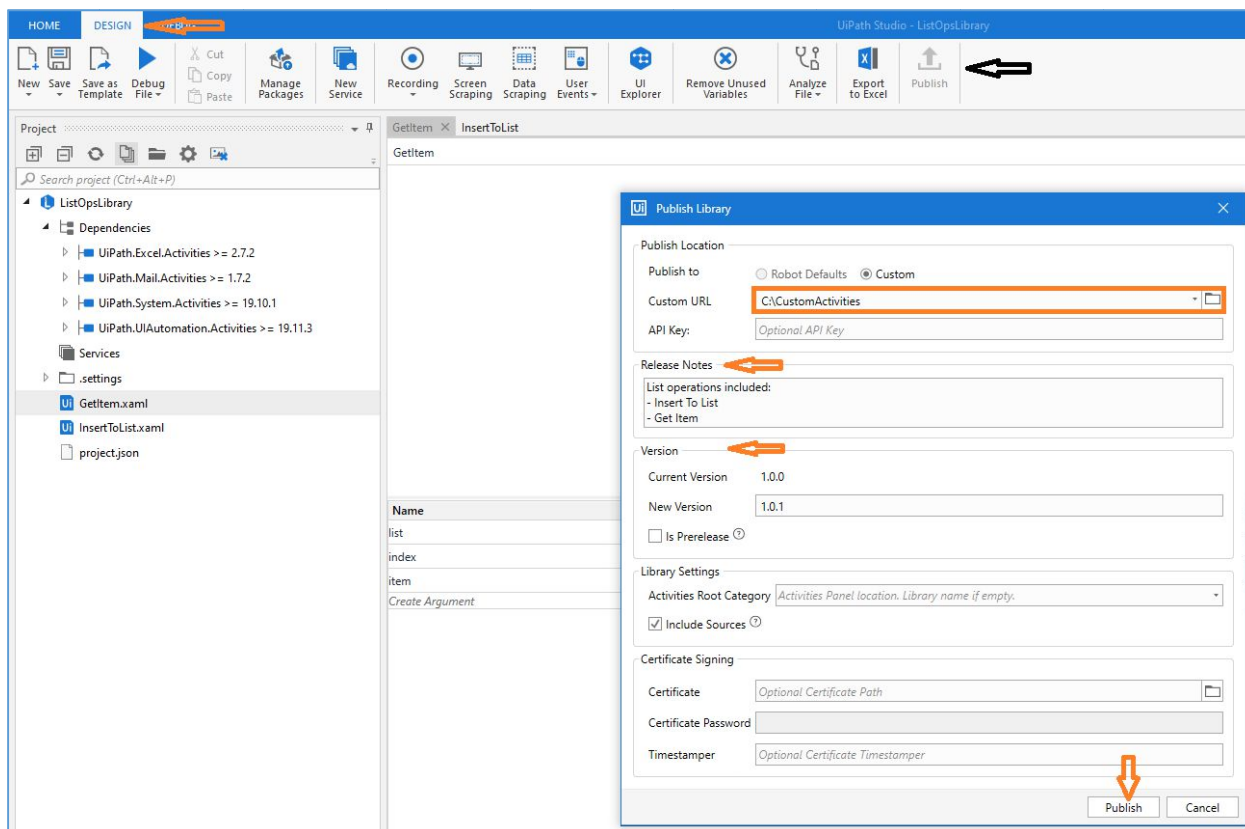


3. Publicarea pachetului de activități personalizate

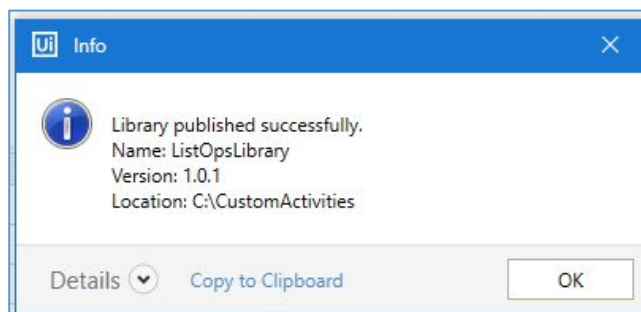
Pas01. Din meniul **Design** se alege opțiunea **Publish**. Fereastra **Publish Library** permite configurările:

- **Custom URL:** folderul care va conține fișierul **.nupkg** generat la finalul procesului de publicare a pachetului de activități. Se va introduce numele folderului de stocare stabilit anterior, adică **c:\CustomActivities**.
- alte informații referitoare la pachetul de activități personalizate, de exemplu, **Version**, **Release Notes**, certificări sau setări suplimentare.

Pas02. Se finalizează publicarea pachetului de activități prin click pe butonul **Publish**. Configurările de bază pentru publicare sunt prezentate în figura de mai jos.



Activitățile publicate pot fi modificate/corectate, ulterior fiind necesară re-publicarea pachetului de activități. În acest fel, numărul asociat versiunii este incrementat automat. Finalizarea cu succes a publicării/re-publicării bibliotecii de activități este indicată prin apariția unui mesaj de confirmare, ca în figura de mai jos.

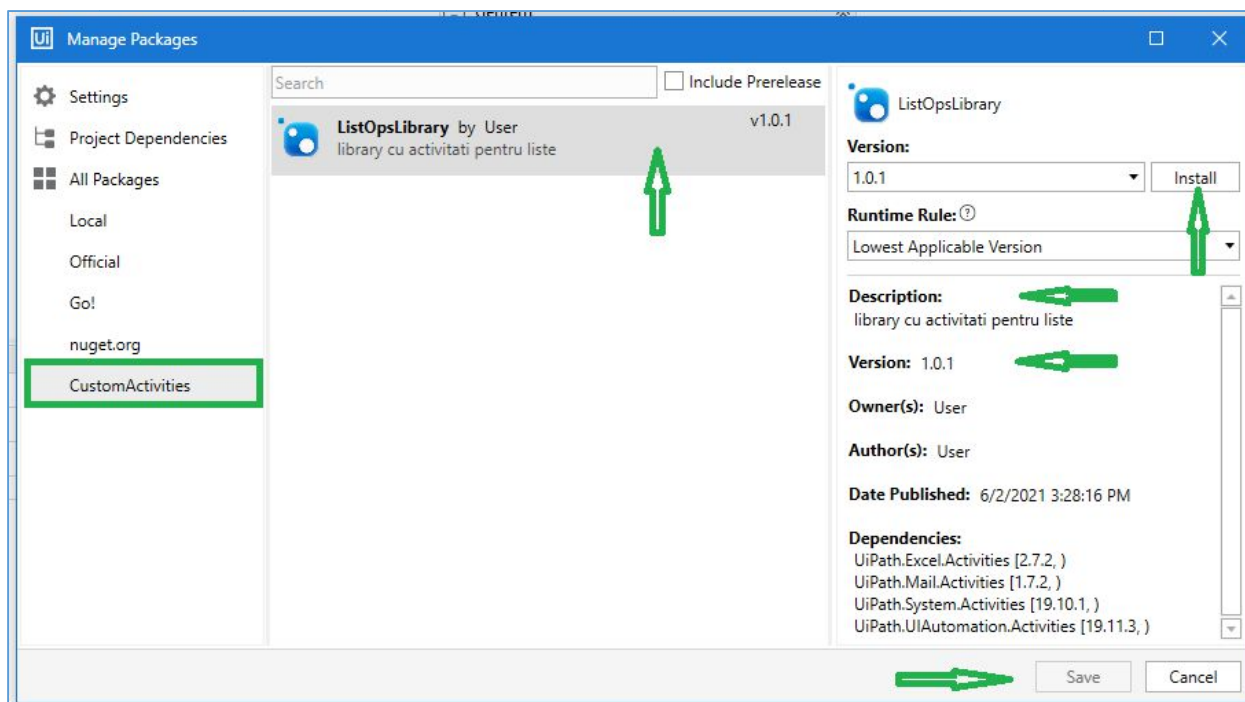


4. Instalarea și utilizarea pachetului de activități definit în UiPath

Procesul de instalare este similar celui aplicat în cazul pachetelor de activități predefinite. Vom presupune că folderul *c:\CustomActivities* conține pachetele de activități publicate care trebuie instalate.

Pas01. Din meniul **Design** se alege opțiunea **Manage Packages** și folderul *CustomActivities* disponibil în lista folderelor cu accesare rapidă. Lista de pachete de activități este actualizată cu toate pachetele de activități distincte existente în folderul selectat.

Pas02. Se selectează pachetul de activități care va fi instalat. Detaliile referitoare la pachetul de activități setate la momentul publicării pachetului devin disponibile, de exemplu descrierea și versiunea, ca în figura de mai jos.

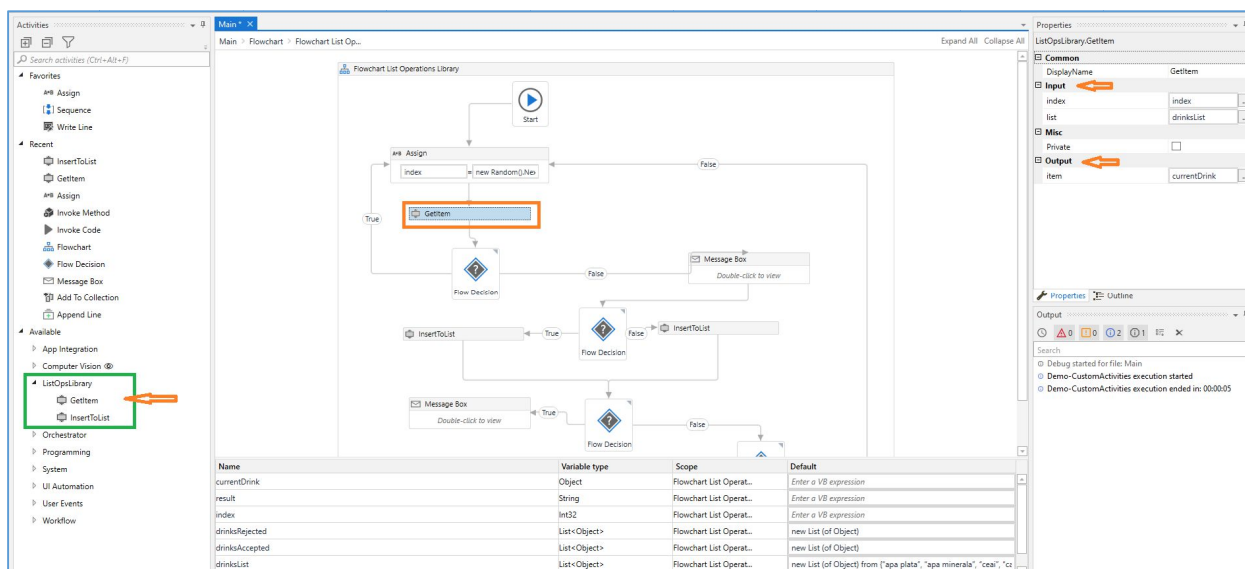


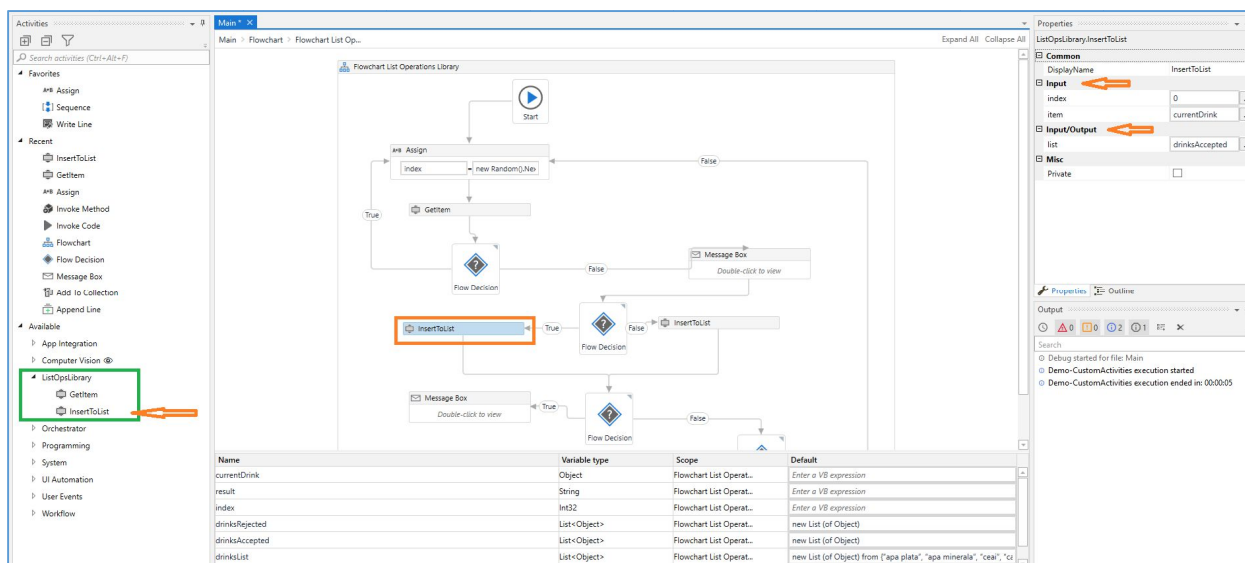
Pachetele de activități instalate pentru care există îmbunătățiri, adică versiuni ulterioare celei instalate, vor fi evidențiate.

Pas03. Se alege opțiunea **Install** și apoi **Save**, ca în figura de mai sus.

Dacă pachetul de activități este deja instalat va apărea opțiunea **Update** și **Save**.

Pas04. După instalare, pachetul de activități este disponibil în lista de activități uzuale folosite de către programator, în cadrul unui grup cu numele pachetului de activități, ca în figurile de mai jos.





După adăugarea unei activități personalizate într-un proiect de tip *Process*, toți parametrii utilizați la definirea activității apar ca atribute ale activității, fiind disponibile în *Properties Panel*.

III. Utilizarea pachetelor de activități dezvoltate în C#

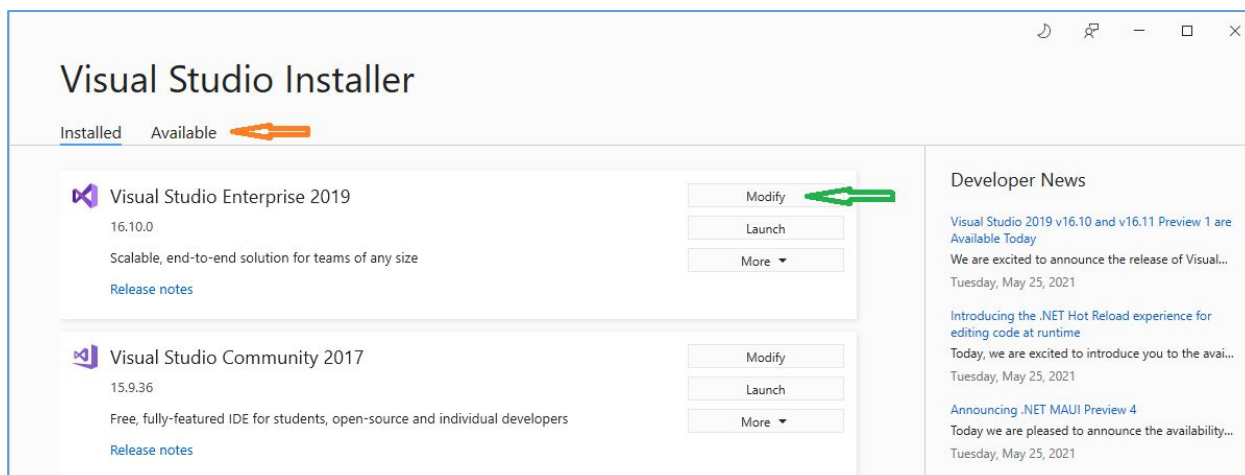
Pentru dezvoltarea de pachete de activități în C#, care ulterior sunt instalate și utilizate în UiPath Studio, este necesară parcurgerea următoarelor etape:

1. Configurarea mediului de dezvoltare Visual Studio;
2. Crearea proiectului C# *Class Library*;
3. Dezvoltarea activităților și crearea pachetului de activități .nupkg
4. Instalarea și utilizarea pachetelor de activități în UiPath Studio

1. Configurarea mediului de dezvoltare Visual Studio

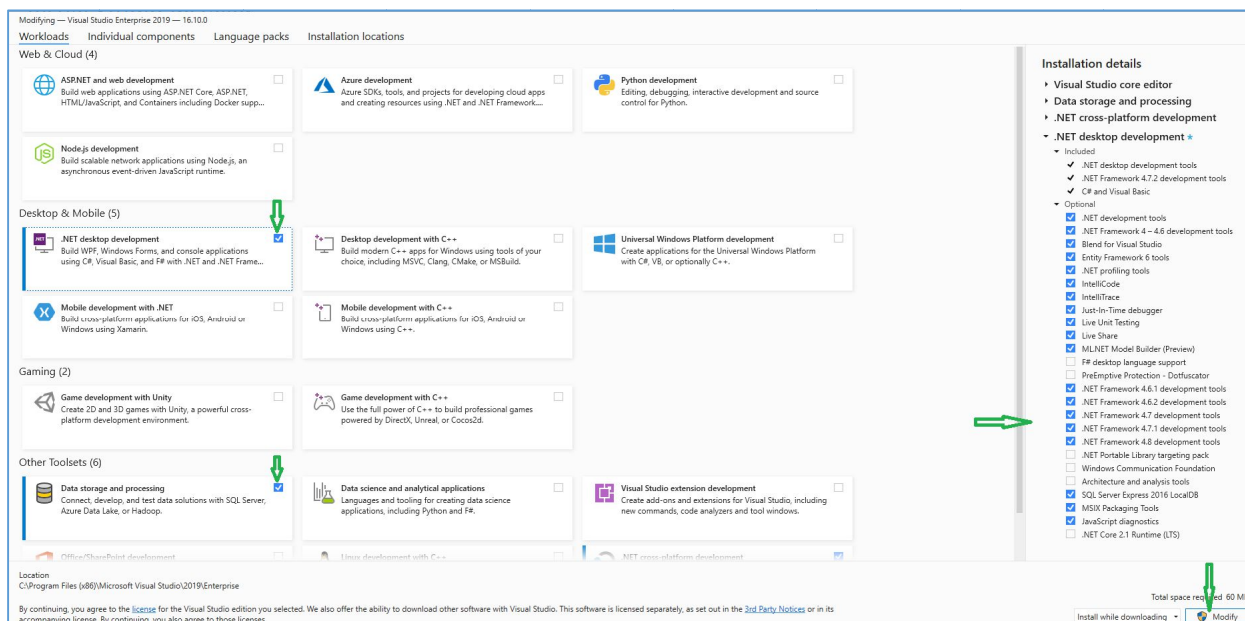
Activitățile se pot crea în Visual Studio folosind un proiect **Class Library**. De asemenea, de dorim să putem genera fișierul .nupkg asociat pachetului de activități dezvoltate în cadrul mediului Visual Studio. Aceste configurări minimale sunt prezentate în pașii de mai jos.

Pas01. La instalarea Visual Studio 2019 (VS 2019), din cadrul aplicației Visual Studio Installer putem configura facilitățile care să fie instalate, folosind opțiunea **Modify**, dacă VS 2019 este instalat anterior, ca în figura de mai jos.



Pas02. Facilitățile necesare pentru dezvoltarea activităților care trebuie adăugate în VS 2019 sunt:

- framework-ul *.NET Framework 4.x* + pentru opțiunea **.NET desktop development**;
- opțiunea **Data storage and processing**, conform figurii de mai jos.



2. Crearea proiectului C# Class Library

Dezvoltarea activităților de către programator în limbajul C# folosind VS 2019 presupune utilizarea unui proiect tip *Class Library (.NET Framework)*. Generarea fișierului .nupkg necesar pentru instalarea pachetului de activități în UIPath Studio se poate urmând următoarele strategii:

Strategia1. Class Library (.Net Framework) + NuGet Package Explorer:

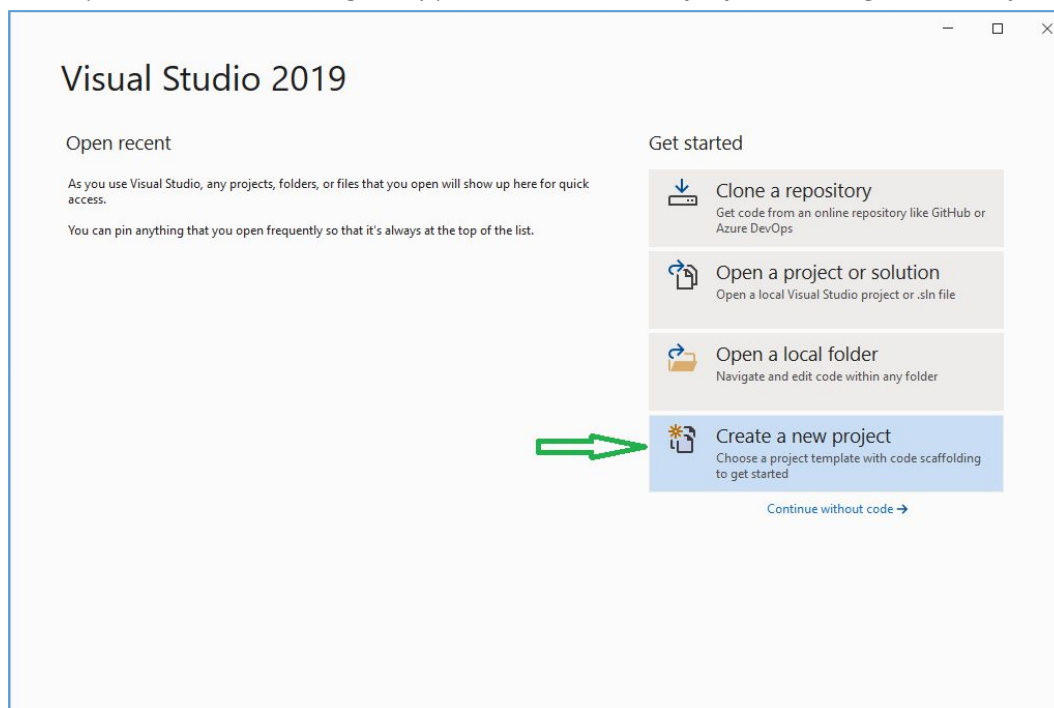
- utilizarea unui proiect **Class Library (.NET Framework)** și obținerea fișierului .dll;
- utilizarea aplicației **NuGet Package Explorer** care va crea fișierul .nupkg pe baza fișierului .dll obținut anterior;

Strategia2. Class library (.NET Core/.Net Standard) + Class Library (.Net Framework):

- utilizarea unui proiect **Class library (.NET Core/.Net Standard)** care permite generarea fișierelor .nupkg;
- alterarea fișierului .csproj care transforma proiectul în **Class Library (.NET Framework)** și permite obținerea fișierului .nupkg necesar.

Mai jos sunt descriși pașii corespunzători *Strategiei2* descrise mai sus, care utilizează doar VS 2019 pentru obținerea fișierelor .dll și .nupkg.

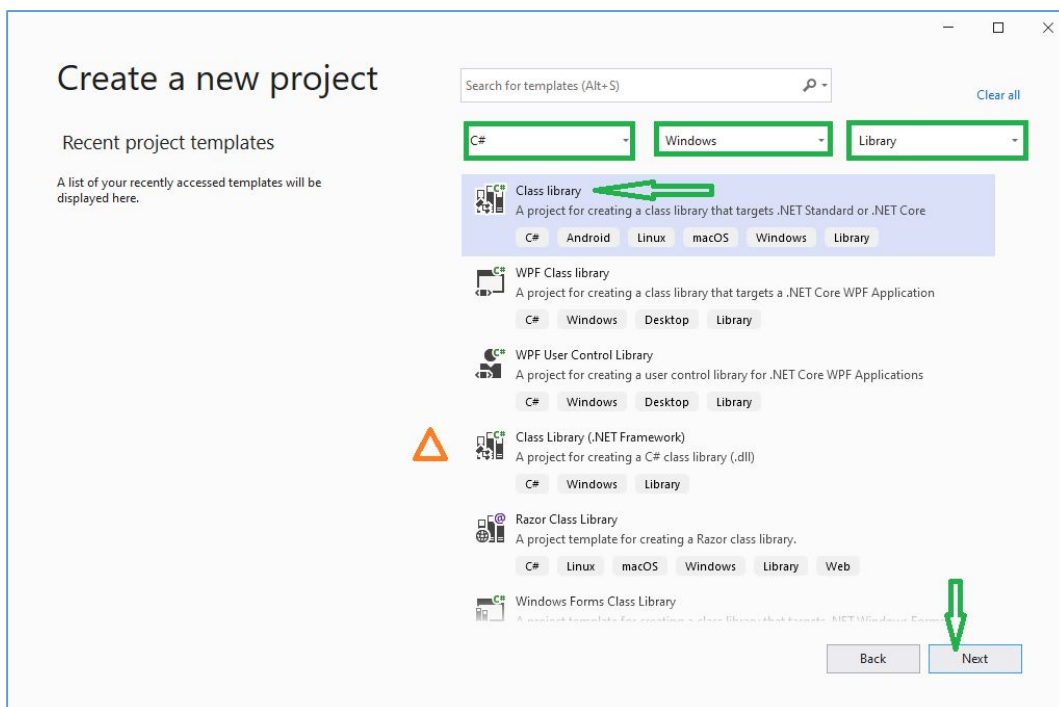
Pas01. La inițializarea VS 2019, alegem opțiunea **Create a new project**, ca în figura de mai jos.



Pas02. Se aleg următorii parametri de creare ai proiectului în VS Studio:

- **Language:** C#;
- **Platform:** Windows;
- **Type of the project:** Library

De asemenea, din lista cu tipuri de proiecte *Library* se alege **Class library**, pentru crearea proiectelor *.NET Core* și *.NET Standard*. Apoi, se alege opțiunea **Next**, ca în figura de mai jos.

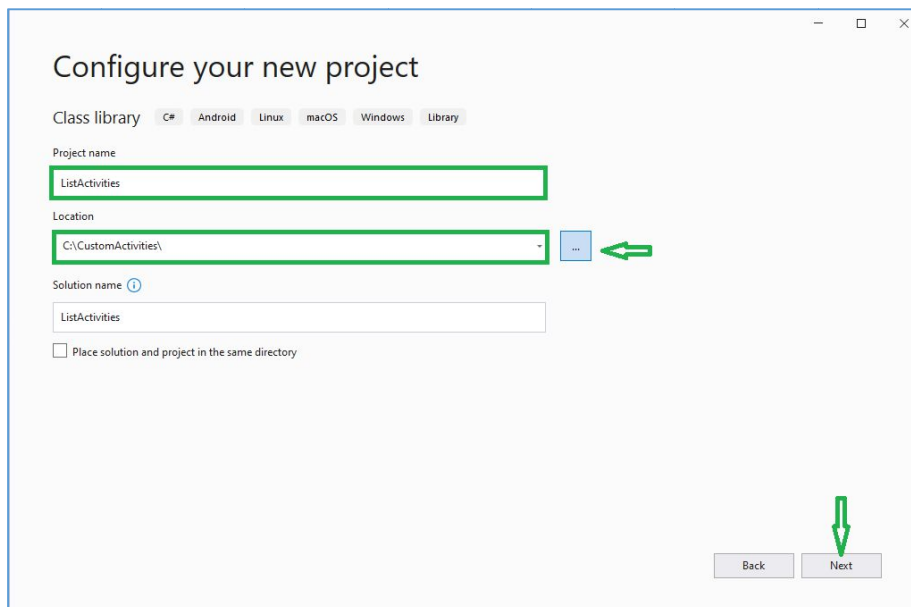


Tipul de proiect **Class Library (.NET Framework)**, marcat în figura de mai sus, este tipul de proiect care se creează inițial dacă alegem să aplicăm *Strategia1*.

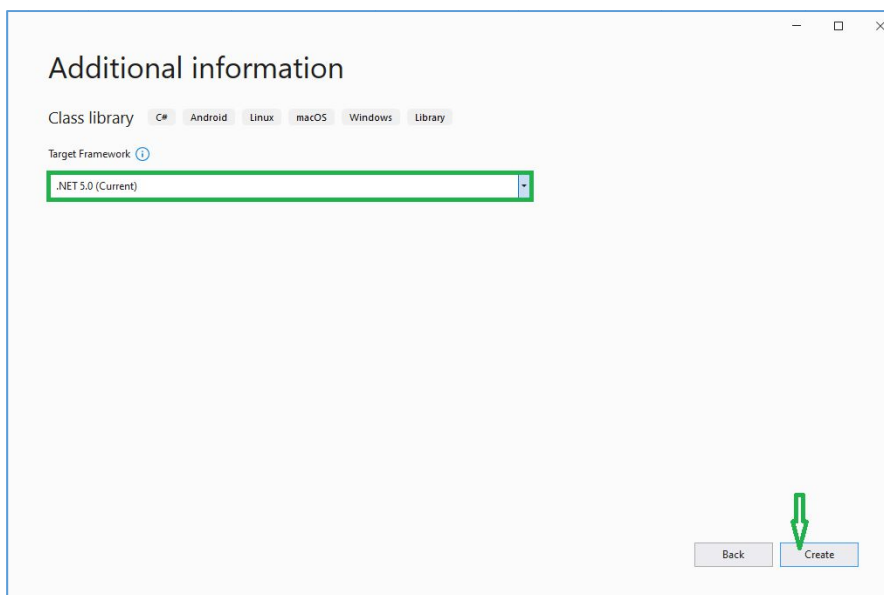
Pas03. Se precizează celelalte opțiuni de configurare ale proiectului:

- **Project Name:** ListActivities;
- **Location:** C:\CustomActivities

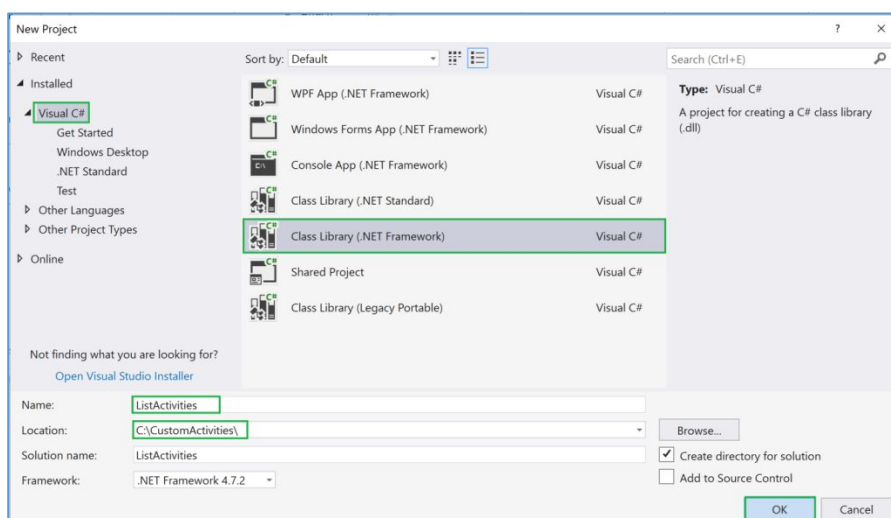
Apoi se alege opțiunea **Next**, ca în figura de mai jos.



Pas04. Se alege opțiunea **.NET 4.x +** care corespunde framework-ului folosit și apoi se finalizează crearea proiectului prin opțiunea **Create**.

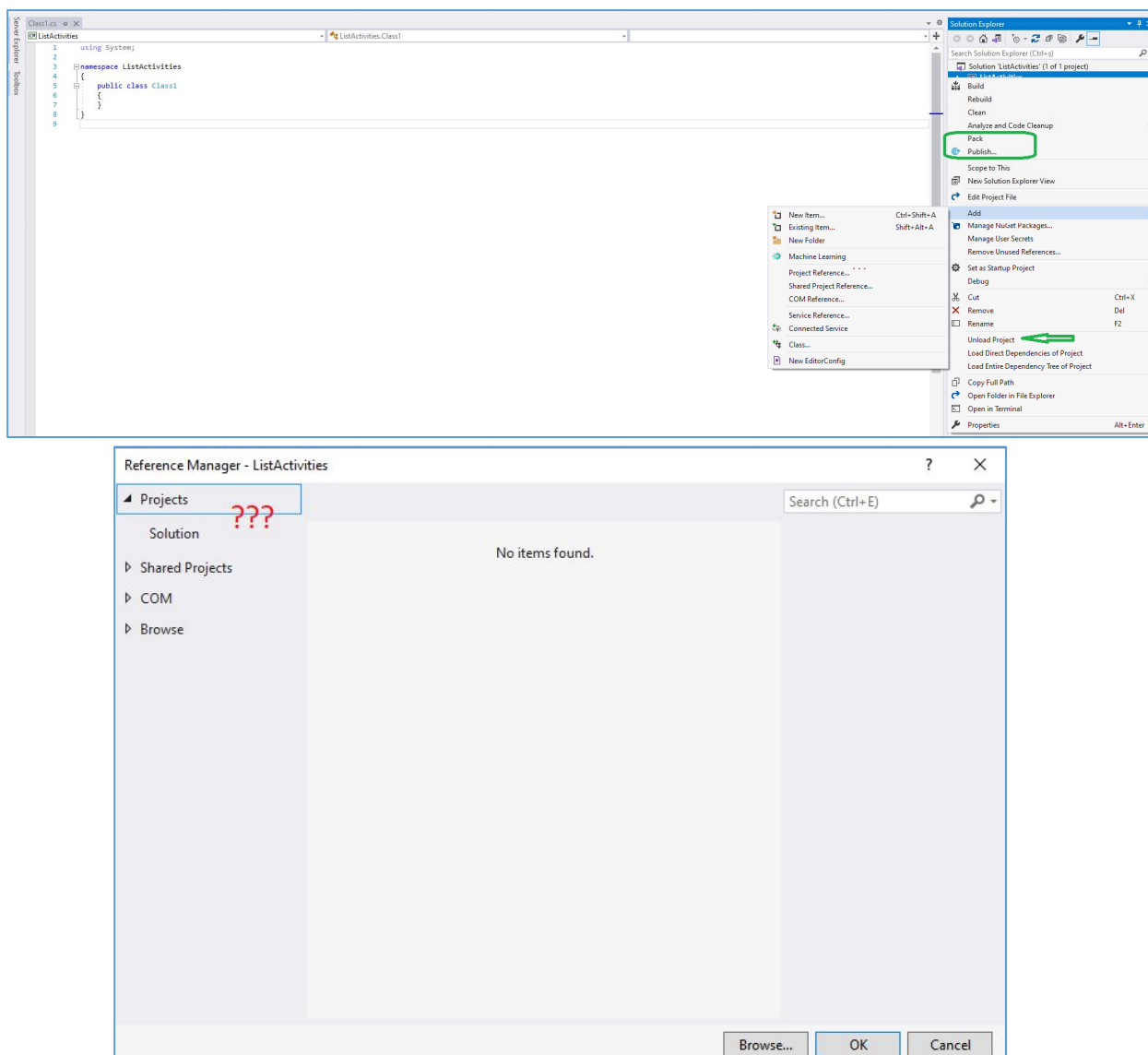


În cazul în care se folosește VS 2017, crearea proiectului **Class Library (.Net Framework)** se realizează ca în figura de mai jos, fiind permise toate configurările proiectului într-o singură fereastră.



Pas05. Proiectul de tip **Class library (.NET Core/.NET Standard)** permite imediată a crearea fișierului de tip .nupkg, dar pentru crearea activităților avem nevoie de utilizăm alte componente care nu sunt disponibile în cadrul unui proiect class library **.NET Core/.NET Standard**, ci într-un proiect **.NET Framework**.

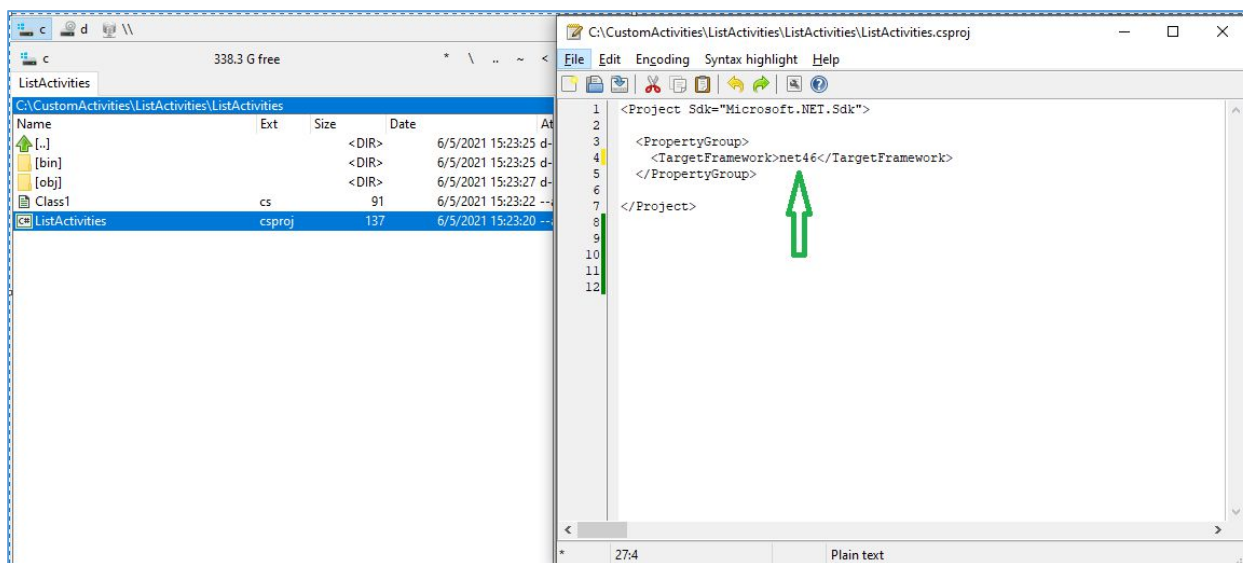
Acest lucru este vizibil în la accesarea ferestrei pentru gestionarea referințelor prin click dreapta pe numele proiectului în *Solution Explorer Panel*, alegerea opțiunii **Add** și apoi a opțiunii **Project References**, ca în figurile de mai jos.



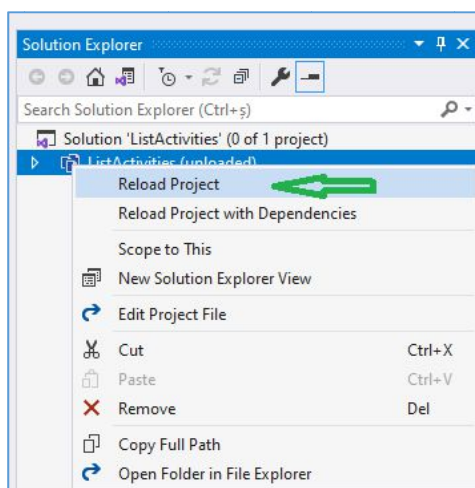
Putem avea acces la facilitățile dorite transformând proiectul deja creat într-unul **.NET Framework**.

Etape de transformare:

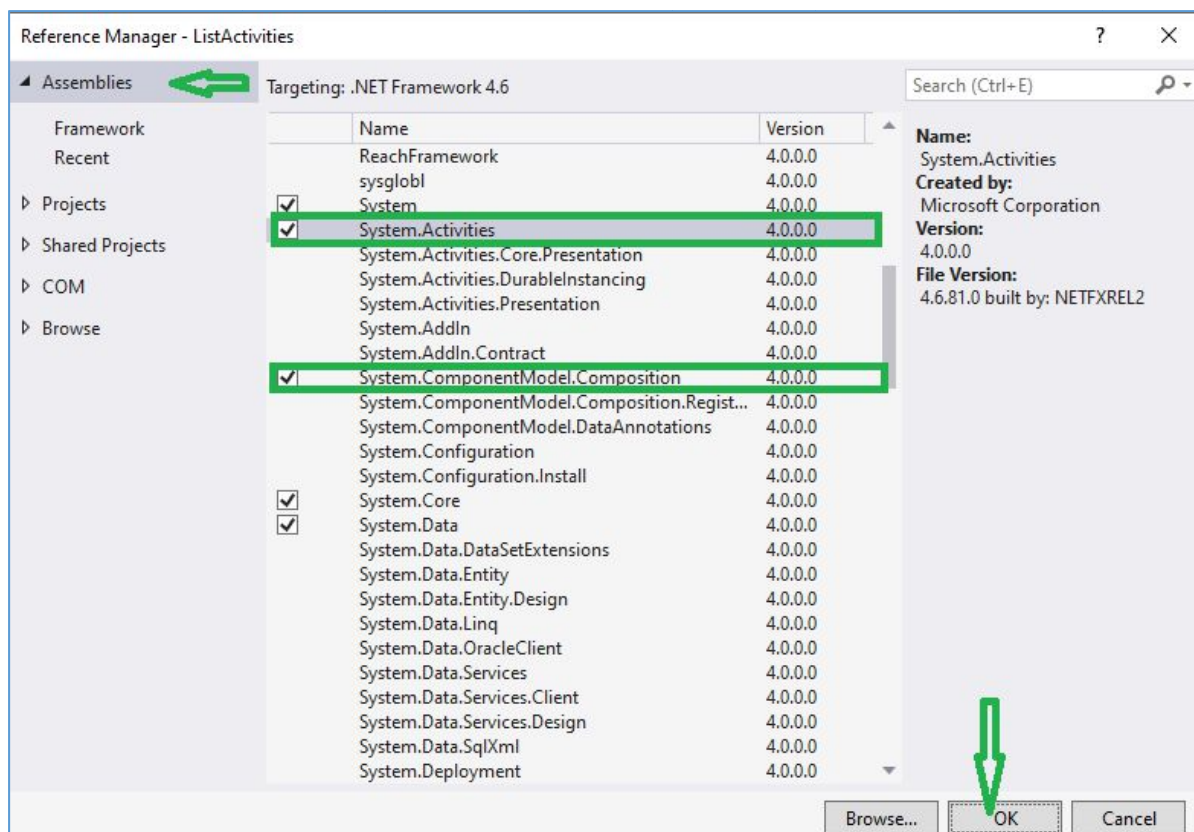
- Realizăm operația *unload* la nivelul proiectului, prin click dreapta pe numele proiectului în *Solution Explorer Panel* și alegerea opțiunii **Unload Project**, ca în figura de penultima figură.
- Edităm fișierul *.csproj* disponibil în folderul solution al proiectului. Pentru proiectul **ListActivities**, acest fișier este **ListActivities.csproj**, disponibil în folderul **ListActivities\ListActivities**. În acest fișier vom modifica versiunea framework-ului folosit, din **net5.0** în **net46**, ca în figura de mai jos. Apoi salvăm modificările realizate în acest fișier.



- c. În VS 2019, realizăm operația încărcare a proiectului, prin click dreapta pe numele proiectului în *Solution Explorer Panel* și alegerea opțiunii **Reload Project**, ca în figura de mai jos.



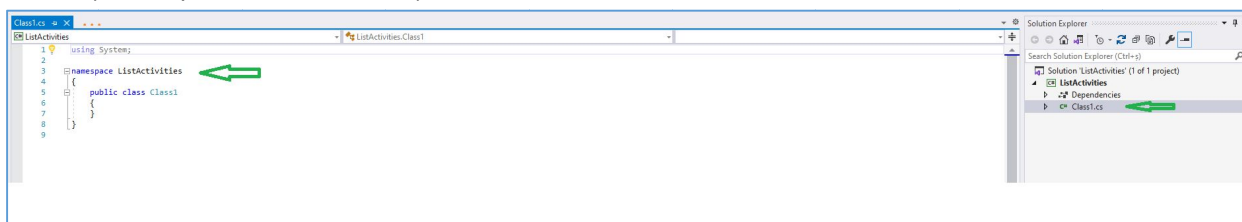
- d. Accesăm din nou fereastra pentru gestionarea referințelor prin click dreapta pe numele proiectului în *Solution Explorer Panel*, alegerea opțiunii **Add** și apoi a opțiunii **Project References**. Rezultatul este vizibil în fereastra de mai jos, unde facilitățile de care avem nevoie vor disponibile în secțiunea *Assemblies*.



e. Din secțiunea *Assemblies* vom selecta dependențele:

- **System.Activities**
- **System.ComponentModel.Composition**, ca în figura de mai sus.

Apoi alegem opțiunea **OK** pentru a fi include în proiect. În continuare, vom putea descrie activitățile în cadrul proiectului configurat complet. Activitățile vor face parte din namespace-ul (pachetul) *ListActivities*. Numele fișierului este *Class1.cs*, ca în figura de mai jos, dar poate fi modificat. În cadrul namespace-ului vom putea adăuga clase (similare clasei *Class1*) care reprezintă activitățile din pachetul de activități dezvoltate.



3. Dezvoltarea activităților și crearea pachetului de activități .nupkg

Pentru dezvoltarea activităților se va ține cont de următoarele aspecte:

- Pentru fiecare activitate care se creează se adaugă o clasă distinctă. Astfel, vom avea clasele *GetItem* și *InsertToList* adăugate în namespace-ul *ListActivities*. Acesta va da și numele pachetului de activități vizibil în UiPath Studio.
- Fiecare dintre cele două clase adăugate este o clasă derivată din clasa *CodeActivity* și namespace-ul **System.Activity**, indicat anterior ca dependență.

- Fiecare clasă adăugată va avea attribute (care indică starea activității) și metode (care indică comportamentul/acțiunile activității).
 - attributele pot fi de tip *input*, *output* sau *input/output* și vor fi vizibile în *Properties Panel* în UiPath Studio atunci când vom selecta activitatea pe care o dezvoltăm acum;
 - attributele definite pot fi asociate unor categorii implicite, cum ar fi: **Input**, **Output**, **Input/Output**, **Common**, **Result** sau **Misc**. De asemenea, putem crea propriile categorii de parametri ai activității care vor apărea în *Properties Panel* în UiPath Studio.
 - metoda care trebuie adăugată implicit este metoda **Execute** din clasa moștenită, pe care o va suprascrie și nu e va modifica semnatura. Pe lângă această metodă, putem adăuga alte metode care contribuie la definirea corectă și completă a acțiunilor activității definite de această clasă.
- În fișele care conțin definirea claselor asociate activităților e vor include namespace-urile asociate dependențelor. Astfel vom avea:

```
using System.Activities;  
using System.ComponentModel;
```

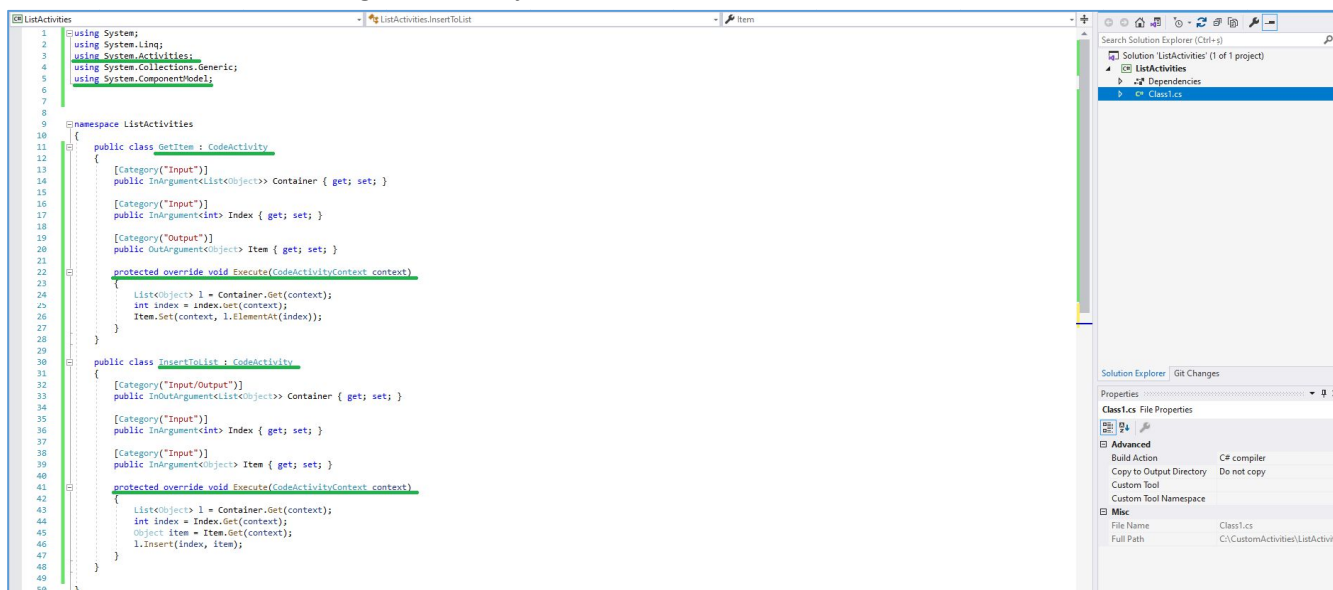
Mai jos sunt descrise clasele asociate activităților **GetItem** și **InsertToList** definite în C#.

```
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)); //Item.Set(context, l[index]);  
    }  
}
```

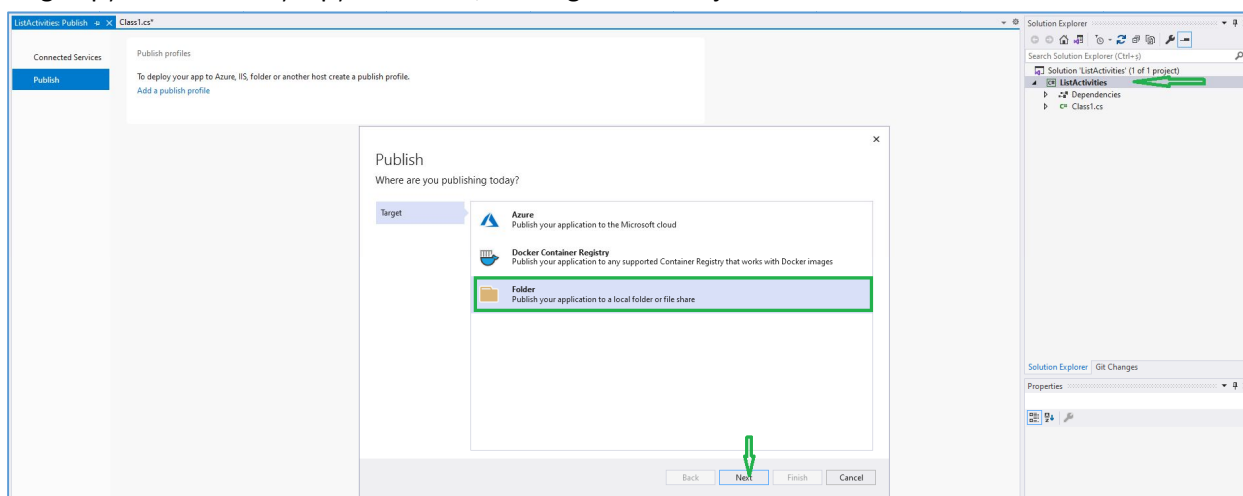
```
public class InsertToList : CodeActivity  
{  
    [Category("Input/Output")]  
    public InOutArgument<List<Object>> Container { get; set; }  
  
    [Category("Input")]  
    public InArgument<int> Index { get; set; }  
  
    [Category("Input")]  
    public InArgument<Object> Item { get; set;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

}

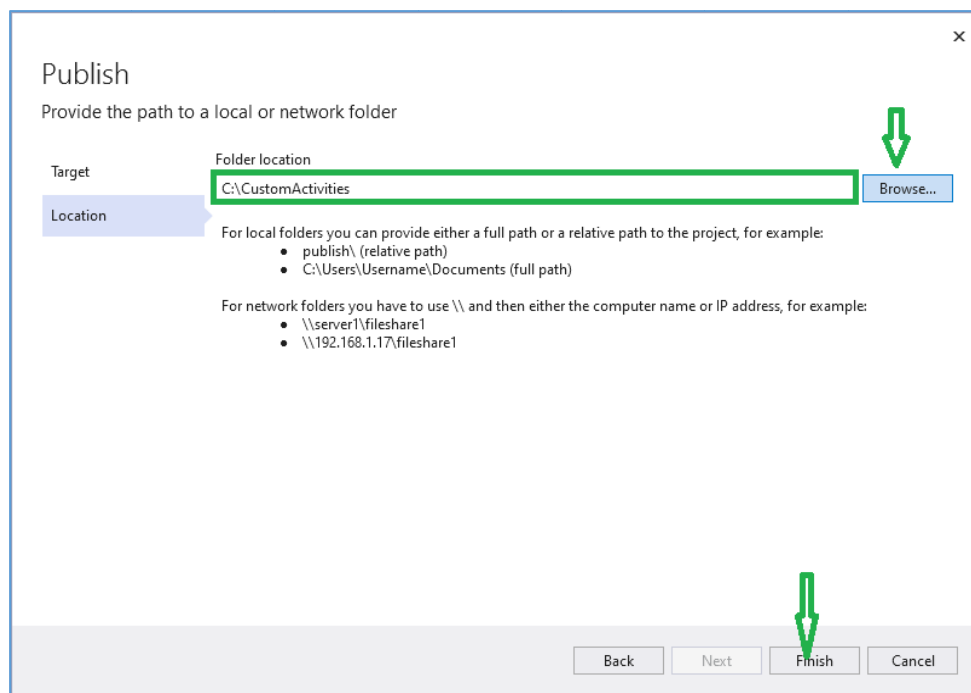
Codul sursă va arăta ca în imaginea de mai jos.



Pentru crearea pachetului .nupkg se alege folderul în care se va publica pachetul de activități. Astfel, prin click dreapta pe numele proiectului în *Solution Explorer*, se alege opțiunea **Publish....** Apoi se alege opțiunea **Folder** și opțiunea **Next**, ca în figura de mai jos.

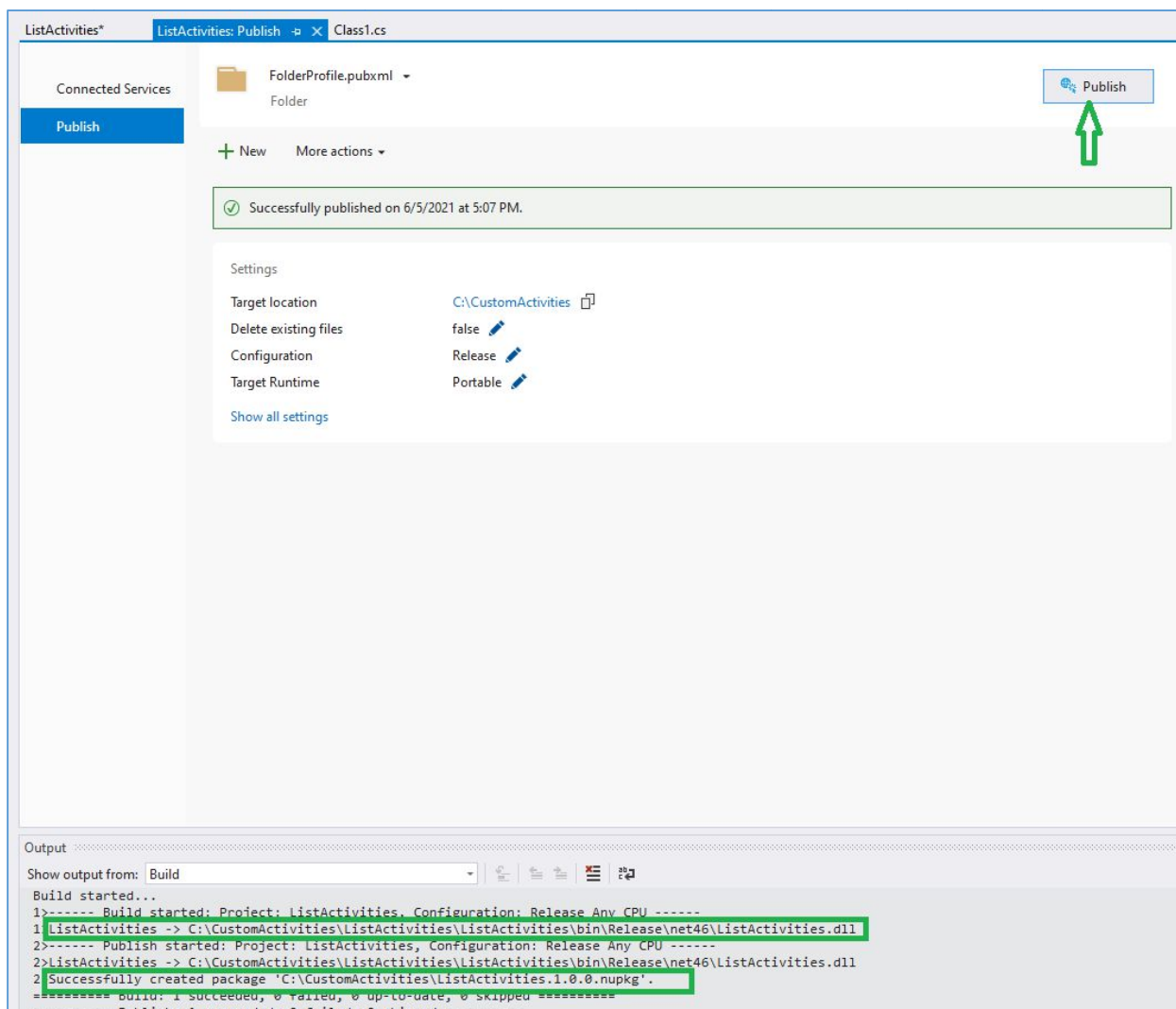


Se alege folderul *c:\CustomActivities* prin opțiunea **Browse**, iar apoi se salvează configurarea prin **Finish**.

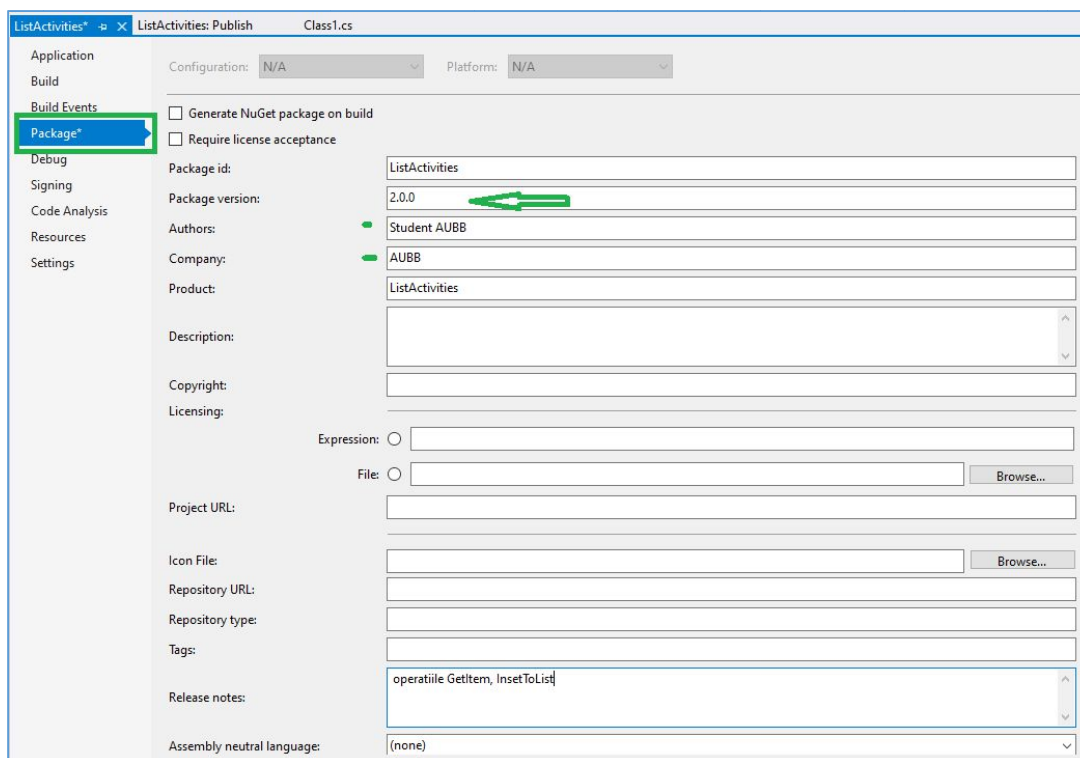


Prin opțiunea Publish se realizează două acțiuni:

- compilarea și obținerea fișierului `ListActivities.dll`;
- crearea pachetului de activități pentru a utilizat în afara VS 2019, obținând fișierul `ListActivities.1.0.0.nupkg`, stocat în folderul setat anterior, conform figurii de mai jos.



Versionarea pachetului de activități este importantă, deoarece permite activităților dezvoltate să fie îmbunătățite sau corectate. Pentru obține o nouă versiune, înainte de publicare se modifică versiunea, noului pachet, prin click dreapta pe numele proiectului în *Solution Explorer* și alegerea opțiunea **<numele_proiectului> Properties**. În fereastra de proprietăți a proiectului, în secțiunea **Package** se pot realiza diferite configurări pentru pachetul publicat, ca în imaginea de mai jos.

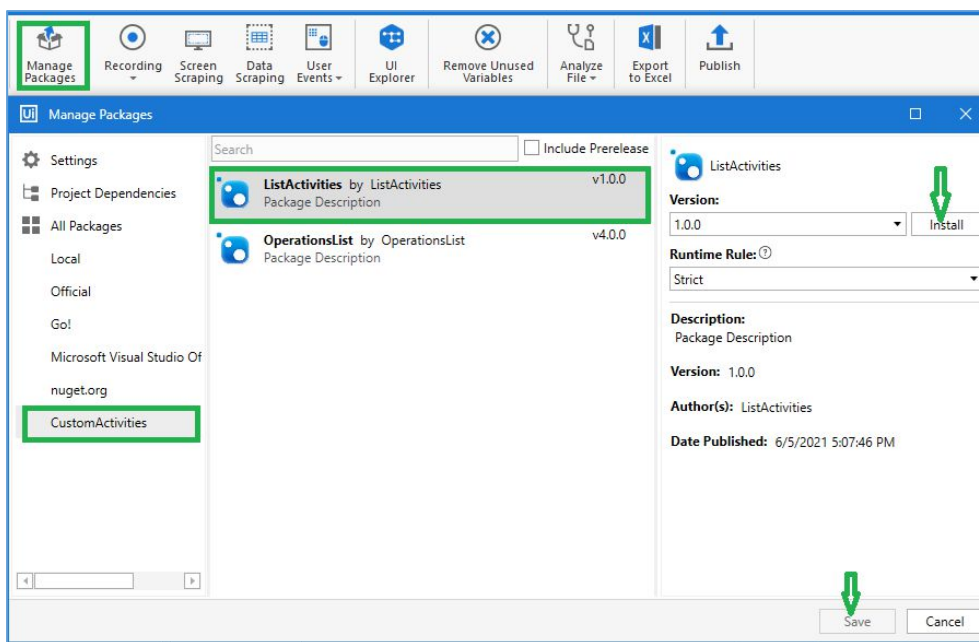


4. Instalarea și utilizarea pachetelor de activități în UiPath Studio

Procesul de instalare este similar celui aplicat în cazul pachetelor de activități predefinite. Vom presupune că folderul `c:\CustomActivities` conține pachetele de activități publicate care trebuie instalate.

Pas01. Din meniul **Design** se alege opțiunea **Manage Packages** și folderul *CustomActivities* disponibil în lista folderelor cu accesare rapidă. Lista de pachete de activități este actualizată cu toate pachetele de activități distincte existente în folderul selectat.

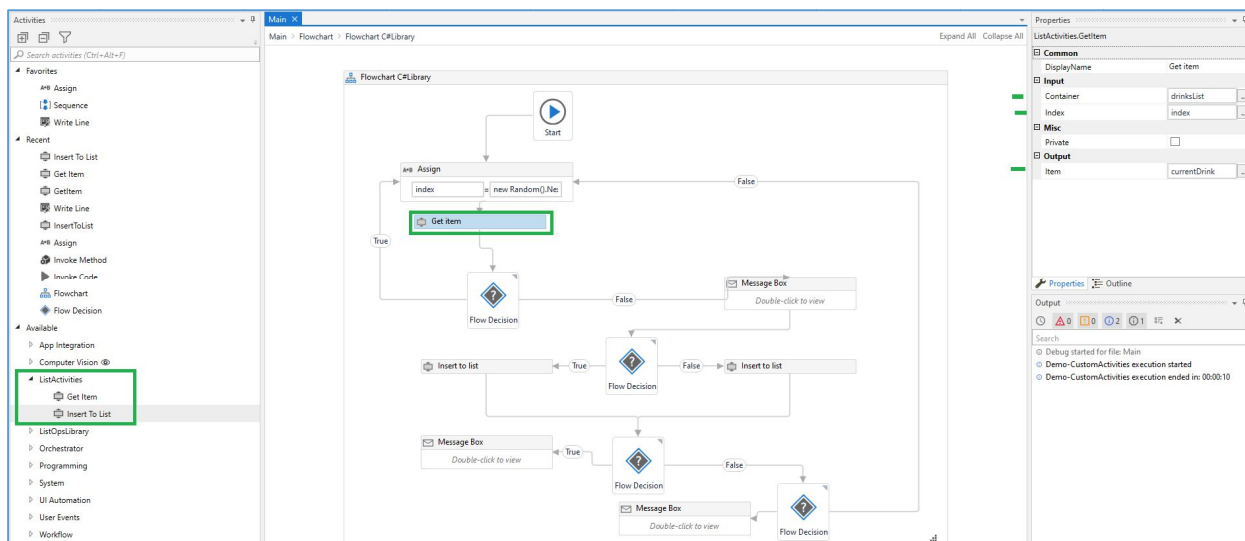
Pas02. Se selectează pachetul de activități **ListActivities** dezvoltat în C#. Detaliile referitoare la pachetul de activități setate la momentul publicării pachetului devin disponibile, de exemplu descrierea și versiunea, ca în figura de mai jos.

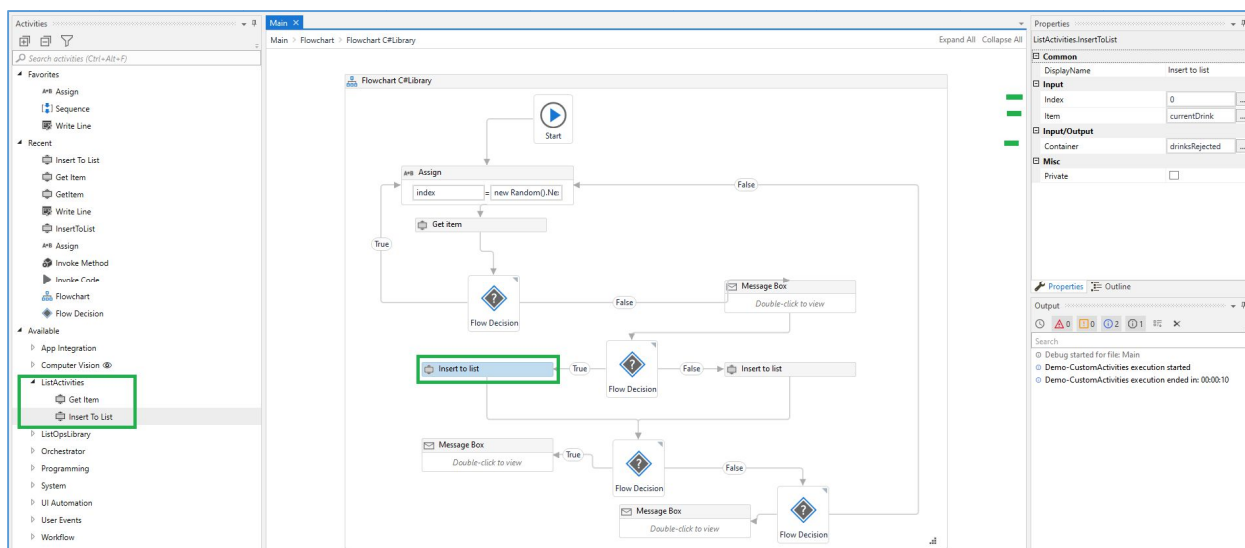


Pas03. Se alege opțiunea **Install** și apoi **Save**, ca în figura de mai sus.

Dacă pachetul de activități este deja instalat va apărea opțiunea **Update** și **Save**.

Pas04. După instalare, pachetul de activități este disponibil în lista de activități uzuale folosite de către programator, în cadrul unui grup cu numele pachetului de activități, ca în figurile de mai jos.





Alte resurse pentru documentare

- UiPath Docs **InvokeCode** activity - <https://docs.uipath.com/activities/docs/invoke-code>
- UiPath Docs Proiect Template - <https://docs.uipath.com/studio/docs/creating-basic-library>
- UiPath Docs Crearea activitatilor proprii – <https://docs.uipath.com/activities/docs/creating-a-custom-activity>
- UiPath Docs Crearea activitatilor in C# - <https://docs.uipath.com/marketplace/docs/how-to-create-activities>