UiPath Activities Manual
SharePoint Custom Activities Package
-custom activities-

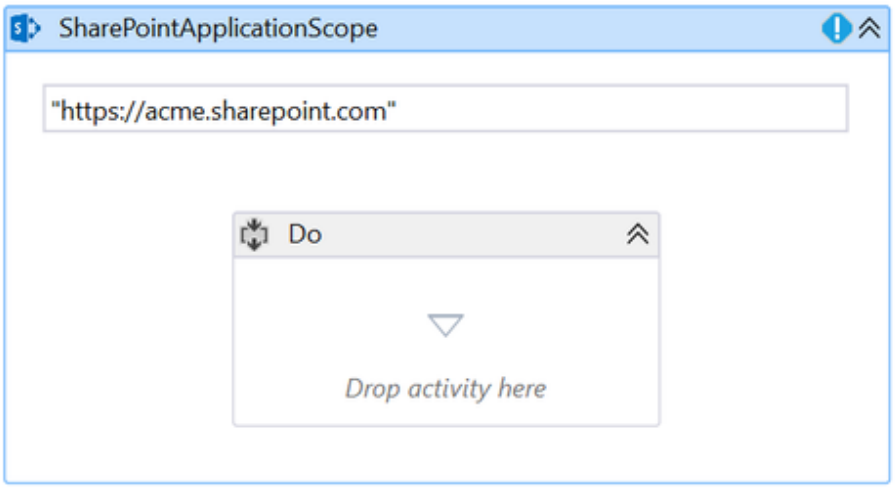
Table of Contents

Objective

- Provide an integration between UiPath and SharePoint that permits the usage of the key functionalities of SharePoint that are most likely to be used in an automation.
- Allow document and folder management for Libraries.
- Allow CRUD functionalities on SharePoint Lists.
- Allow the creation/deletion of groups and the administration of the permissions assigned to them and of the users inside.
- Grouping the queries sent to the SharePoint server (whenever possible) so that we can do many requests in a relatively short amount of time.

Activities

1. SharePoint Activity Scope



1.1. Description

This activity will be used as a container for all the other activities in the package. Its main purpose is to perform the authentication with the SharePoint site and to organize the queries sent by the other SharePoint activities (whether they are grouped together, or they are sent individually).

1.2. Parameters

- **URL:** The URL of the SharePoint site where we want to perform all our queries
    - All the SharePoint activities contained inside will be executed at the level of this site
- **UserName** & **Password:** The credentials that should be used to connect to the site **(If the account used does not have the necessary permissions, some of the activities will throw exceptions)**
    - **SharePointInstanceType:** A choice between **OnPremises** and **Online** (default). Based on the type of SharePoint instance you have, choose the appropriate value (for Office 365 select the Online option).
- **QueryGrouping:** Option which the user can enable in order to specify that all the SharePoint queries done inside this activity scope should be executed at the end
    - This option allows us to more efficiently execute many requests in a relatively short amount of time (For example if you want to add 100 new items to a list, you can group all the queries in order to achieve better performance)
    - **Observation!** Using this option will only let you use the following activities inside this Scope: **AddListItem, AddPermission, RemovePermission, AddUserToGroup, CreateGroup, DeleteGroup, RemoveUserFromGroup.**

- **ClientContext:** an **out** argument of type Microsoft.SharePoint.Client.ClientContext that can be used to send requests to the SharePoint site in **Invoke Code** or **Invoke Method** activities.

2. SharePoint.Activities.Lists

A set of activities which provide the basic CRUD functionality for a SharePoint List:

- AddListItem
- ReadListItems
- DeleteListItems
- UpdateListItems

2.1. AddListItem

This activity adds a **single** item to a SharePoint List located on the SharePoint Site specified in the parent SharePoint Scope Activity. Since adding items to a list can be a repetitive task, this activity can be used together with the **QueryGrouping** feature, in order to improve performance.

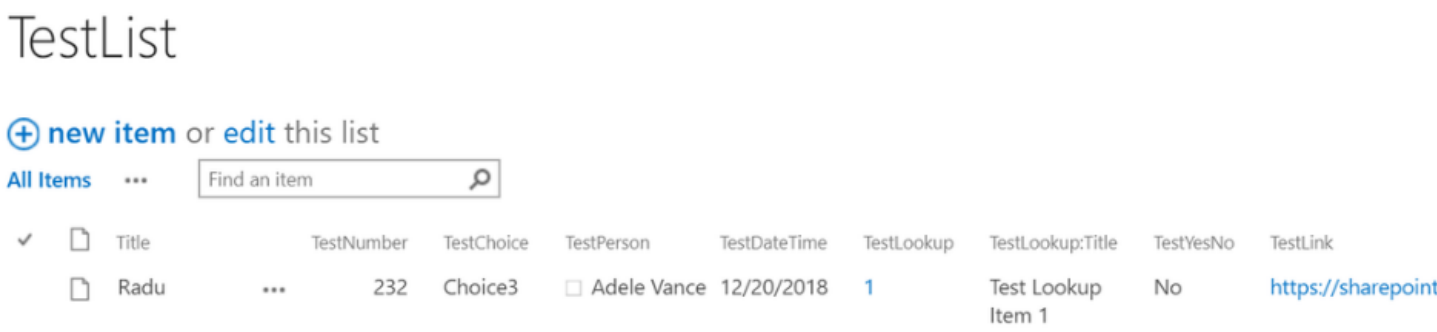
2.1.1. Parameters

- **ListName:** The name of the SharePoint List where we want to add the item
- **PropertiesToAdd:** A **Dictionary<string, object>** object which should contain the values this list item should have. For each element in the dictionary, the **Key** is a string which represents the internal name a field has inside the SharePoint list and the **Value** is an object representing the actual value the newly added item will have inside that field.

- **AddedItemID:** An **out** argument representing the ID of the newly added item.
    - **If the QueryGrouping option is enabled, this argument will be null!**

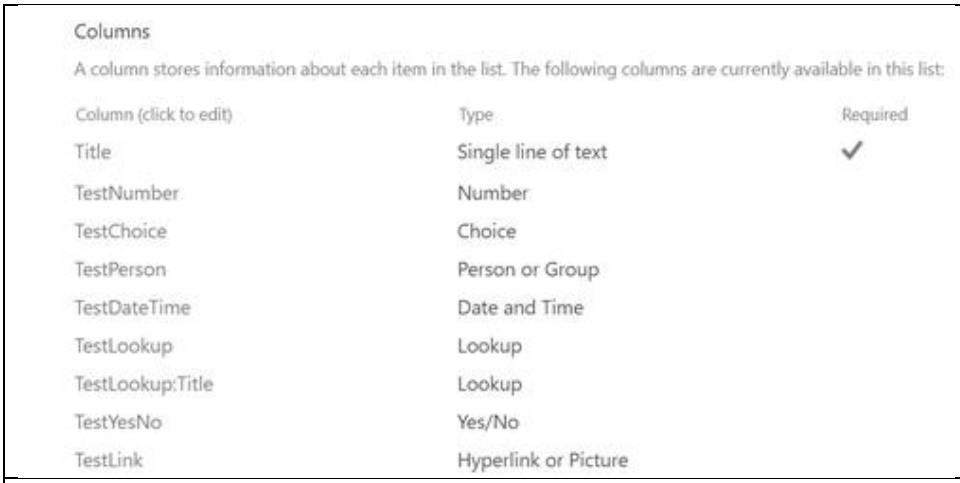
2.1.2. Example

A SharePoint List can have a multitude of data types so adding an item inside a list that makes use of all these data types can be challenging. In order to understand how we can do it, let's try to add an item to the following list:



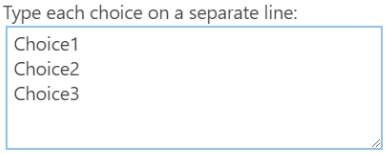
Now, lets take a closer look at my list fields:

Inside TestList we have the following data types



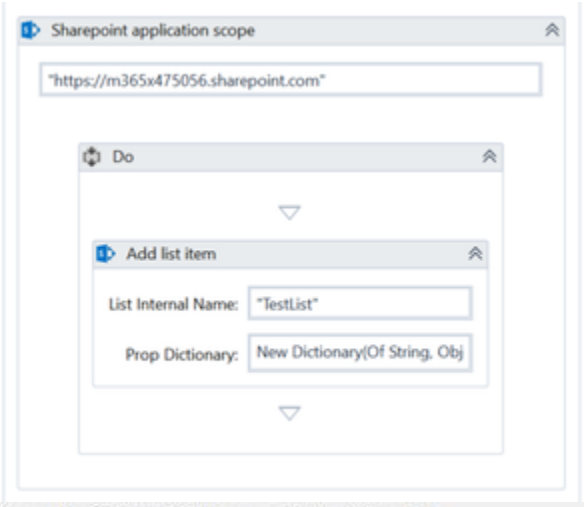
This is a detailed explanation for all of them

- **Title** is of type **Text**, so it is basically a plain string
- **TestNumber** is a field of type **Number**, so in order to add a value here we will use an Int or a double value
- **TestChoice** is of type **Choice**, so it is basically a string picked out of a predetermined set of values. If you go to your list's settings and open the field, you will find the options there. In my case, they are:



- **TestPerson** is a column which simply hold a reference to a person, so we can set it using the Persons ID and a **Microsoft.SharePoint.Client.FieldUserValue.** The ID can be obtained using the **GetUserInformation** activity in this package.
- **TestDateTime** is of type **Date and Time**, so we can add a value here by simply passing a datetime formated as a string
- **TestLookup** is of type **Lookup** which is basically the way in which SharePoint implements associations (one-to-one, one-to-many and even many-to-many) between items in different lists. In order to add a value here, we need to pass the **ID** of the item we want to reference and wrapping it inside an **Microsoft.SharePoint.Client.FieldLookupValue** object. Just go to the list your lookup column is connected to and get the ID of the list item you want to be linked to.
- **TestLookup:Title** is secondary column that is configured to automatically display the title of the value being referenced in the primary lookup column. **So no need to set a value here!**
- **TestYesNo** is basically a boolean value, so we can set it by passing either true or false
- **TestLink** is a Hyperlink so we can set a value to this field by simply passing an url as a string

In order to add a new item here, I will use the Add List Item activity in the following manner:



PropertiesToAdd (Dictionary<String,Object>)

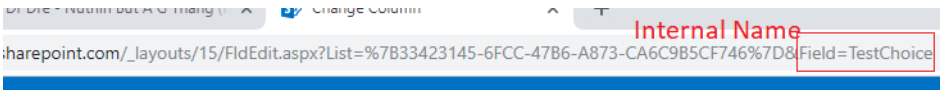
```
New Dictionary(Of String, Object) From {{ "Title", "This is a new item!"},
{ "TestNumber", 222},
{ "TestChoice", "Choice2"},
{ "TestPerson", New Microsoft.SharePoint.Client.FieldUserValue() With { .LookupId = 37 }},
{"TestYesNo",False},
{"TestDateTime", DateTime.Now},
{"TestLookup", New Microsoft.SharePoint.Client.FieldLookupValue() With { .LookupId = 1}},
{"TestLink","https://sharepoint.stackexchange.com"}
}
```

After this workflow is executed, the following item appears in my List:

This is a new item!	...	222	Choice2	Adele Vance	1/28/2019	1	Secondary Item 1	No	https://sharepoint.stackexchange.com

Observation!

One important thing to keep in mind is that in order to reference these fields, we must use their **Internal Name,** not their title (often enough, they are not the same). In order to obtain the internal name of a field open the list settings, click the field and look at the URL of the page, the internal name will be there:



2.2. ReadListItems

This is an activity which uses the Collaborative Application Markup Language (**CAML** for short) to get a set of items from a SharePoint list, filtered on certain criteria.

CAML is complex but it is specifically designed for SharePoint so this means that we can create very powerful and precise queries. Furthermore, we can create these queries using 3rd party tools that generate CAML automatically, so there's no need to be an expert in order to use this activity. A good example of such a tool is SmartCAML, which is free in the Windows store: https://www.microsoft.com/en-us/p/smartcaml/9nn8gjpnxvfg. If you want to learn CAML the hard way, you can check it out here: https://docs.microsoft.com/en-us/sharepoint/dev/schema/introduction-to-collaborative-application-markup-language-caml.

This activity cannot be used with the Query Grouping feature enabled on the parent SharePoint Scope.

2.2.1. Parameters

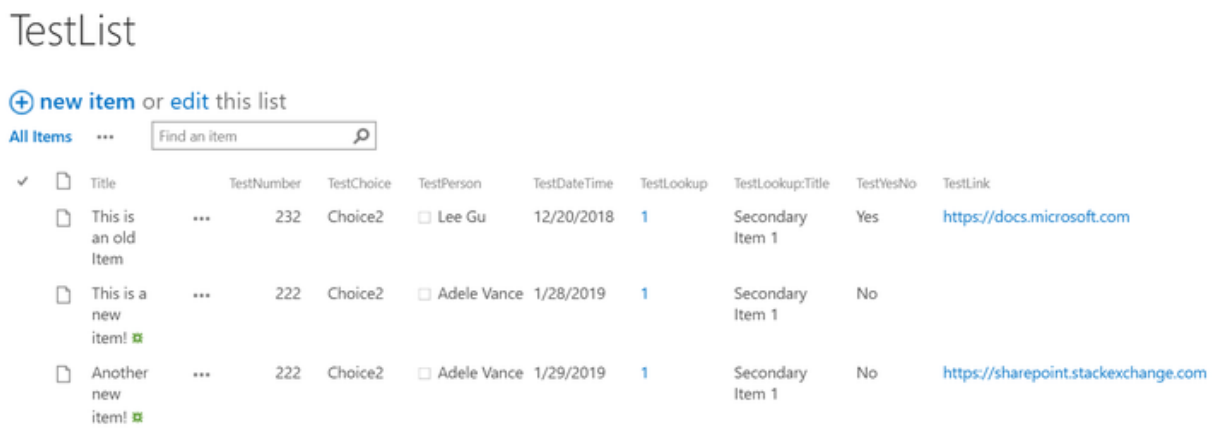
- **ListName:** The name of the SharePoint List where we want to read items from
- **CAMLQuery:** a string containing a valid CAML query that will be used to filter the specified list

- **ResultedItems:** An **out** argument which will contain the returned items filtered using the CAMLQuery. It will be an array of **Dictionary<string, object>** objects.

2.2.2. Example

Let's take the same list that we used for the Create List Item example and try to read some its items.

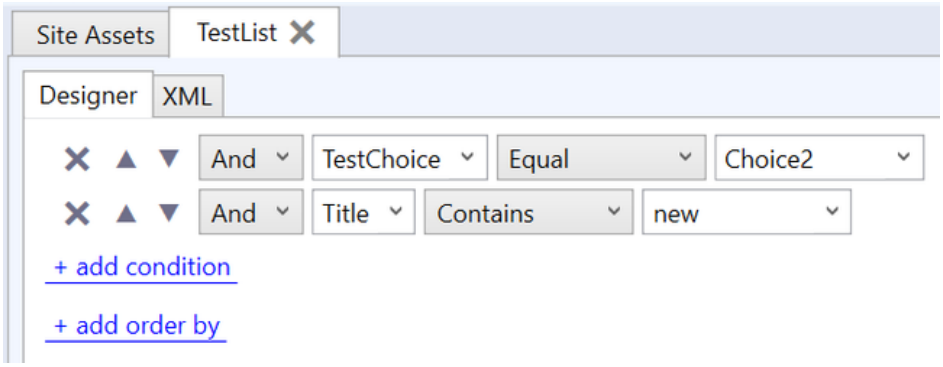
My list currently looks like this:



Let's say that I want to extract all the items whose **Title** contain **"new"** and their **TestChoice** field contains **"Choice2".** I will use my SmartCAML (https://www.microsoft.com/en-

us/p/smartcaml/9nn8gjpnxvfg) to quickly create these filtering options and then transfer them to my SharePoint activity:

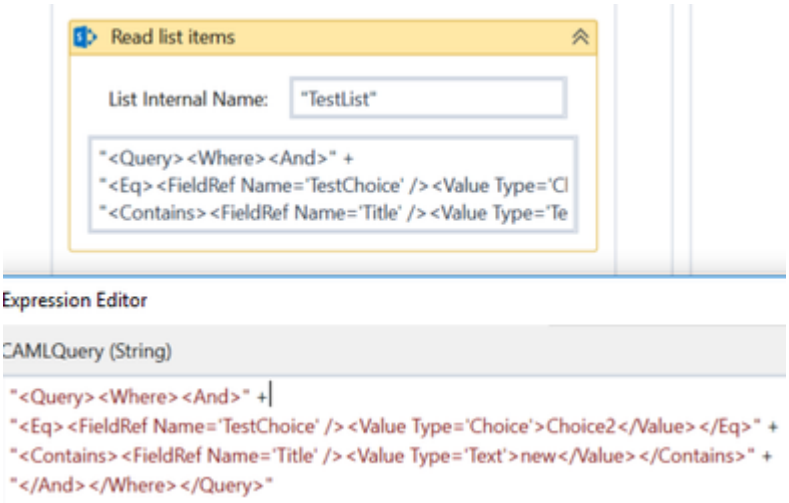
Query Configurations in SmartCAML



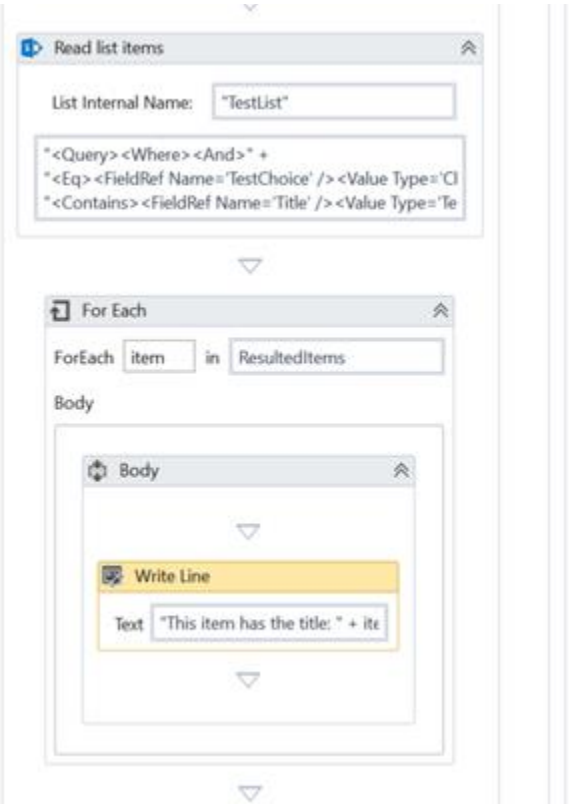
Resulting XML Query

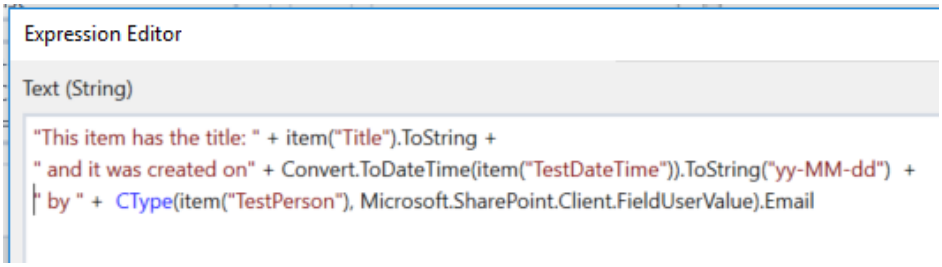
```
<Query>
 <Where>
  <And>
   <Eq>
    <FieldRef Name="TestChoice" />
    <Value Type="Choice">Choice2</Value>
   </Eq>
   <Contains>
    <FieldRef Name="Title" />
    <Value Type="Text">new</Value>
   </Contains>
  </And>
 </Where>
</Query>
```

My query added to my Read List Items activity

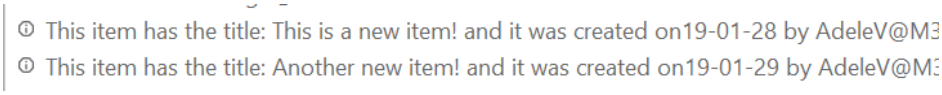


I'll also add a ForEach to loop through all the results and a Writeline activity that will write the information from each item to the Output:





The output will be from the above flow will be:



2.3. DeleteListItems

This is an activity which uses the Collaborative Application Markup Language (**CAML** for short) to delete a set of items from a SharePoint list, filtered on certain criteria. This activity cannot be used with the Query Grouping feature enabled on the parent SharePoint Scope.

For an example on how to use a CAML query, read section 2.2 of this document.

2.3.1. Parameters

- **ListName:** The name of the SharePoint List where we want to delete items from
- **CAMLQuery:** a string containing a valid CAML query that will be used to filter the specified list and retrieve the items that need to be deleted.
- **AddedItemID:** An **out** argument representing the ID of the newly added item.
    - **If the QueryGrouping option is enabled, this argument will be null!**

- **NumberOfRowsAffected:** An **out** argument of type **Int32** which will contain the number of items deleted using the CAMLQuery.

2.4. UpdateListItems

This is an activity which uses the Collaborative Application Markup Language (**CAML** for short) to select a set of items from a SharePoint list, filtered on certain criteria and then update some of their fields using a dictionary containing all the properties to modify. This activity cannot be used with the Query Grouping feature enabled on the parent SharePoint Scope.

2.4.1. Parameters

- **ListName:** The name of the SharePoint List where we want to update the items
- **CAMLQuery:** a string containing a valid CAML query that will be used to filter the specified list and retrieve the items that need to be updated.
- **PropertiesToAdd:** A **Dictionary<string, object>** object which should contain the values we want to update. For each element in the dictionary, the **Key** is a string which

  represents the internal name a field has inside the SharePoint list and the **Value** is an object representing the actual value we will use to update the items.

- **AddedItemID:** An **out** argument representing the ID of the newly added item.
    - **If the QueryGrouping option is enabled, this argument will be null!**

- **NumberOfRowsAffected:** An **out** argument of type **Int32** which will contain the number of items updated using the CAMLQuery.

2.4.2. Example

Let's take the same list that we used for the previous 2 examples and try to update some its items and let's assume that I want to **change the Title of all the items that were created on January the 28th, 2019**. In this case, I will use the **Created** field (this is a field that is managed internally by SharePoint which contains the date on which the item was created and which is added automatically to all the Lists) to create the following CAML query:

CAML Query

```
"<Query><Where><And>" +
"<Geq><FieldRef Name='Created' /><Value Type='DateTime'>2019-01-
28T00:00:00</Value></Geq>" +
"<Leq><FieldRef Name='Created' /><Value Type='DateTime'>2019-01-
28T23:59:59</Value></Leq>" +
"</And></Where></Query>"

//basically, I want all the items older than 2019-01-28T23:59:59 and newer
than 2019-01-28T00:00:00.
```

Since I want to update the title, I will use the following dictionary for the **PropertiesToAdd** dictionary:

CAML Query

```
New Dictionary(Of String, Object) From {{ "Title", "This item is now old."}}
```

Now, all the items from Created on that day will have the title: "This item is now old."

2.5. GetListItemAttachments

This is an activity which gets the attachment names from a list item.

2.5.1. Parameters

- **ListName:** The name of the SharePoint List where we want to update the items
- **ListItemID:** An argument of type **Int** which should contain the ID of the item for which we want to retrieve the Attachment Names.

- **AttachmentNames:** An OutArgument of type **string[]**. Which contains the file names of all the attachments of the current item

Observation

In order to download the attachments, use the **Get File** activity and the following URL: **/Lists/{ListName}/Attachments/{Item ID}/{Attachment name}**

3. SharePoint.Activities.Libraries

A set of activities which can be used for basic operations on files and folders in a SharePoint Library:

- CreateFolder
- Delete
- GetChildrenNames
- GetFile
- UploadFile

- MoveItem
- RenameItem

These activities provide a way of handling files and folders inside a specific SharePoint Site. Generally, we can reference these files and folders by using their relative URL.

These activities are not compatible with the **QueryGrouping** feature; therefore, they cannot be used if this feature is enabled in their SharePoint Scope.

Determining the Relative URL of a resource inside a library

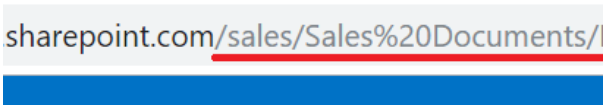
In order to use these activities, we need to determine the relative path of the libraries, folders and files we want to use. Usually these paths have the following format:

/{relative site url}/{library}/{folder path inside the library}/{file name} (**the relative site url** is optional)

OR

/{library}/{folder path inside the library}/{file name} if our site is the root of the site collection

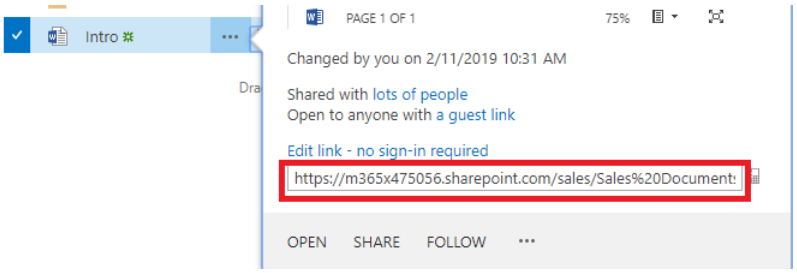
For the **relative site URL** and the **library** part you should navigate to the library page in SharePoint and check the URL:

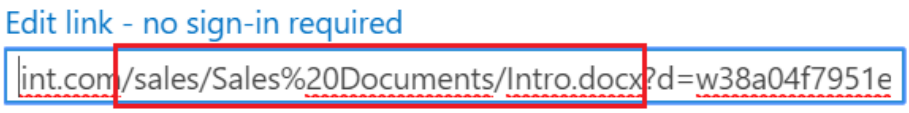


That means that the URL for the library is: **/sales/Sales Documents/** (**the relative site url** is optional)

The **folder Path** can be easily calculated by concatenating the names of all the folder containing our target, starting from the top-most container and going all the way to the direct parent of our current target and simply adding **"/"** between them.

Also, the path of a document/folder can be retrieved by expanding its menu and copying the link and selecting the relative url (from the "sharepoint.com/" until the **"?"** and decoding it)





So, for the above example, I will get the URL: **/sales/Sales Documents/Intro.docx**, if I remove the optional relative site url, I get **/Sales Documents/Intro.docx.**

3.1. CreateFolder

This activity adds a new folder in a library, at the specified path. It creates nested folders as well.

3.1.1. Parameters

- **LibraryName:** the name of the Library in which the folder will be created
- **RelativeUrl:** the url relative to the Library of the new folder. If one folder in the specified URL doesn't exist, the activity creates it.

3.2. Delete

The activity can be used to delete any file or folder in a Library.

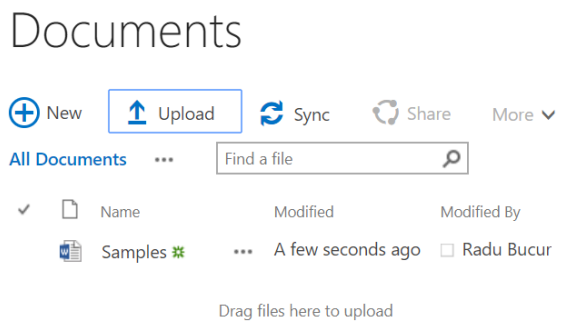
3.2.1. Parameters

- **Library name:** The name of the library where the resource we want to delete is located
- **RelativeUrl:** The relative URL of the file/folder that will be deleted **(relative to the library)**

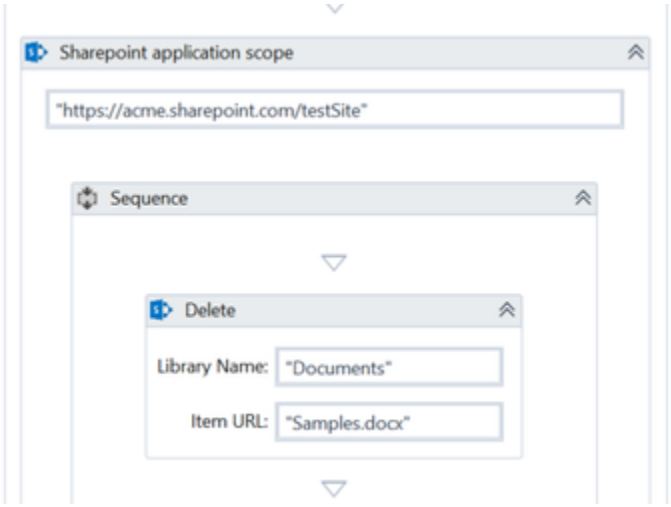
3.2.2. Example

The sample bellow will delete the "Samples.docx" file from the root folder of the "**Documents**" Library.

Here's the document:



And this is the code:



The Item URL relative to the library would be: **“Samples.docx"**

3.3. Get Children Names

The activity returns an array containing the **direct** children names (both folders and files) of a specified folder.

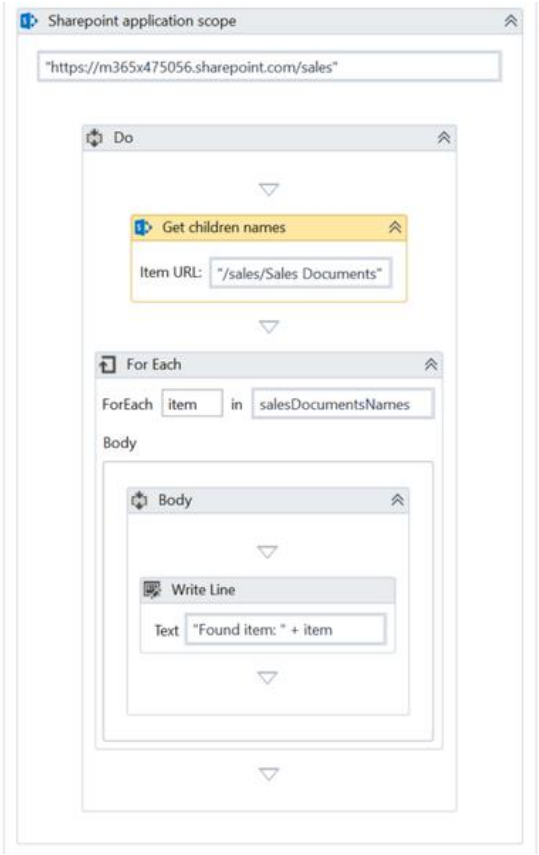
3.3.1. Parameters

- **RelativeUrl:** The relative URL of the parent folder
- **ChildrenNames:** an **OUT** argument of type String[] with the children names

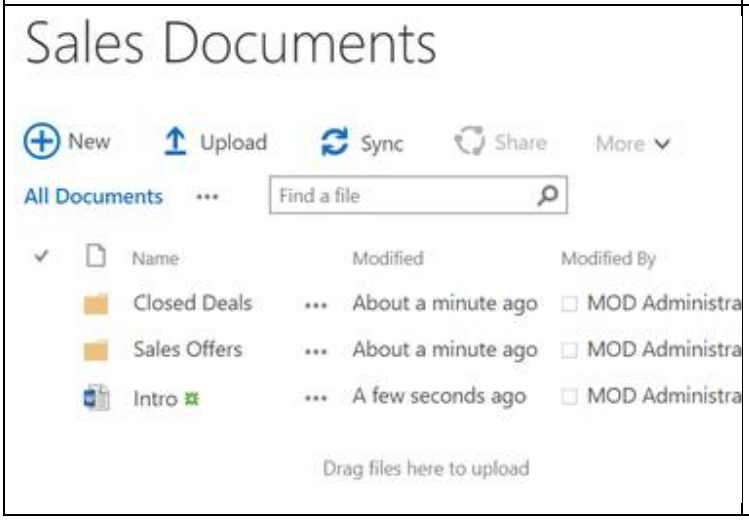
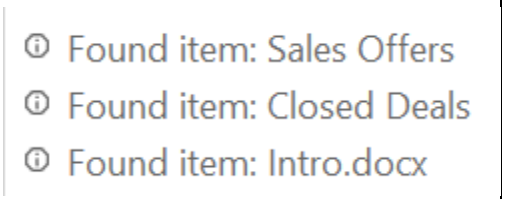
3.3.2. Examples

Suppose we need a list with all direct children of a specific folder in our library **Sales Documents.**

The snippet bellow should do the trick. It will provide us with all the names of the items found in the root folder of the "**Sales Documents**" library:



We have to create a String[] variable (in this case salesDocumentsNames) which will be the output of our activity (as shown in the Properties panel) and will store the array of children names.

Library folder structure	Activity output
	

3.4. Get File

An activity that downloads a file from a specified URL into the mentioned local path.

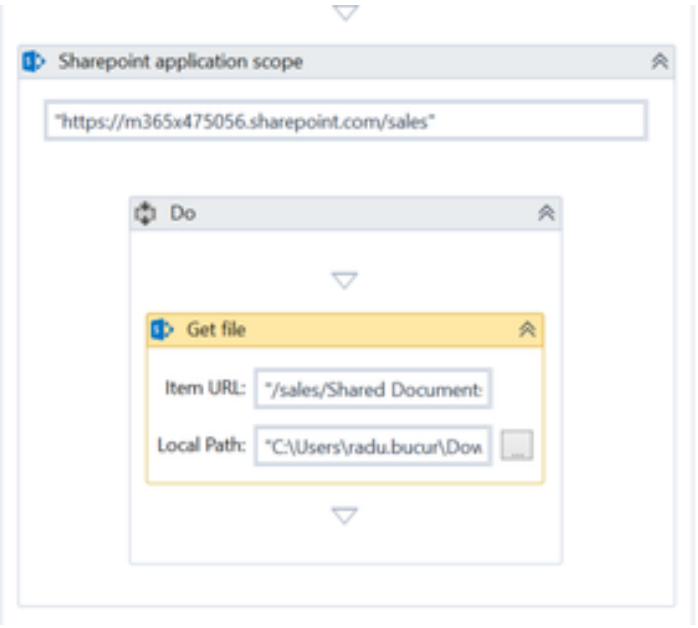
3.4.1. Parameters

- **LocalPath:** Local path where the file will be saved. If it doesn't contain the name that the file will have locally, the name the file has on SharePoint will be used
- **RelativeUrl:** The relative URL of the file which will be saved

3.4.2. Example

Here is a sample of using the GetFile activity to download the "Expenses.xlsx" file from the "TestFolder" folder of the Documents library.

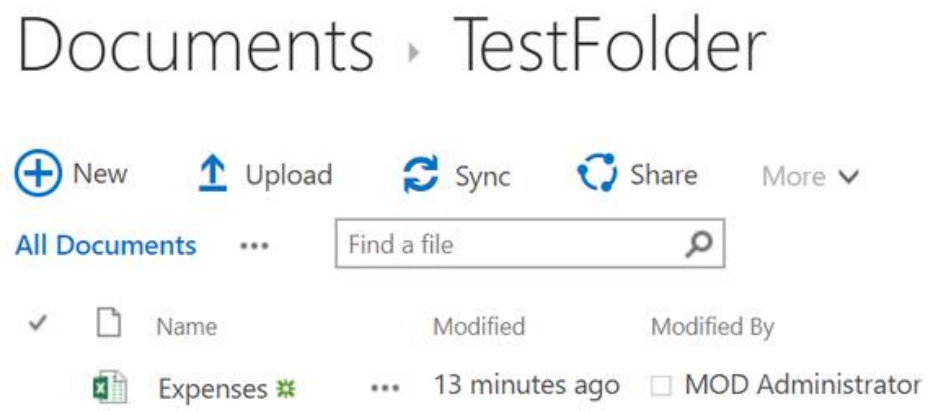
The snippet that does this job it would be:



\

The SharePoint Item URL is: **"/Shared Documents/TestFolder/Expenses.xlsx".** The local path is my Downloads folder.

From SharePoint:



Into the local path:

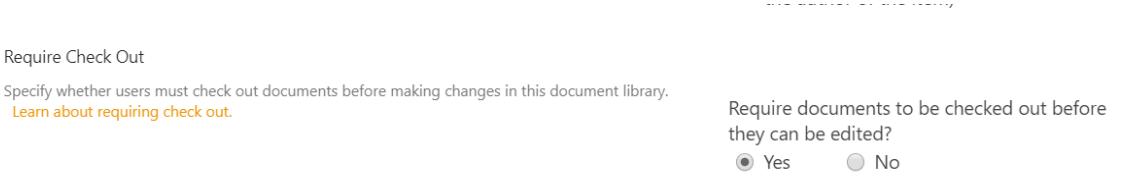
3.5. Upload File

An activity that uploads a file at a specified URL in a Library, from the mentioned local path and adds a set of field values to it.

This will upload the file in the exact path specified in the Item Url. If we don't specify the name the file will have in SharePoint, it will be given the name it initially had locally.

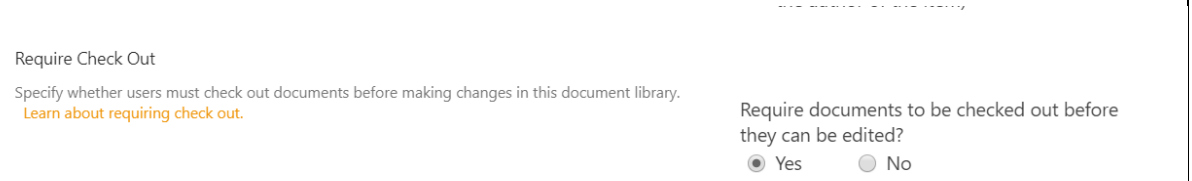
3.5.1. Parameters

- **LocalPath:** The current local path and name of the file to upload
- **RelativeUrl:** The relative URL where the file will be uploaded and its name
- **PropertiesToAdd:** An argument of type **Dictionary<string, object>** that represents a collection of values to be added to the field values of the library for the newly added document. For each element in the dictionary, the **Key** is a string which represents the internal name a field has inside the SharePoint list and the **Value** is an object representing the actual value the newly added document will have inside that field.
- **AllowOverwrite:** Checkbox which specifies whether or not we should let the robot override an existing file with the same Relative URL. Enabled by Default.
- **CheckOutFileBeforeOverwrite:** If there's already a file with the same name at the specified SharePoint location, having this checkbox checked will tell the activity to first check out the file before we overwrite it. Otherwise, no effect.
- **CheckInFileAfterCreation:** If this checkbox is checked, then the robot will attempt to check in the file after it has been uploaded to the server. If the file is not checked out after the upload, no effect



Observations:

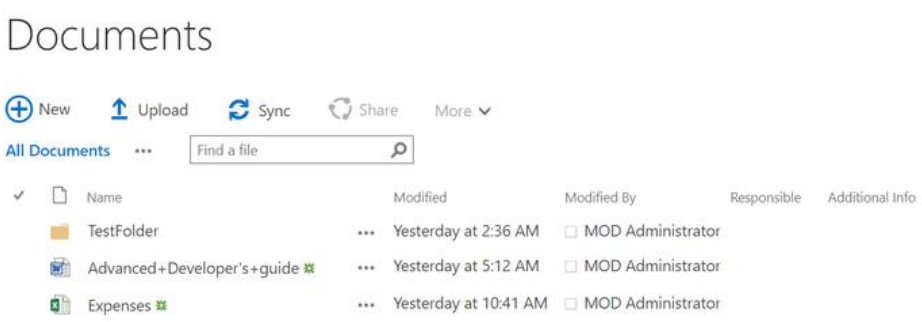
- If an in-depth example is needed on how to add metadata to a Library Item, check the example provided for the **Add List Item** activity.
- If the metadata needs to be edited the **Update List Items** Activity can be used!
- he "**Require Check Out**" option (located in the Versioning Section of the Library Settings) will not let users modify files unless they are checked out. Additionally, because of this option, any new files uploaded to be checked out by default. In order to mitigate this setting, you can enable the properties: **CheckInFileAfterCreation** and **CheckOutFileBeforeOverwrite**



- **CheckInFileAfterCreation** and **CheckOutFileBeforeOverwrite** will cause one extra query to be made, so the activity will be a bit slower. Do not use them unless necessary.

3.5.2. Example

Let's assume that my Documents library has a text field called **Additional Info** and another field called **Responsible** where the current person that is responsible for the document is added:

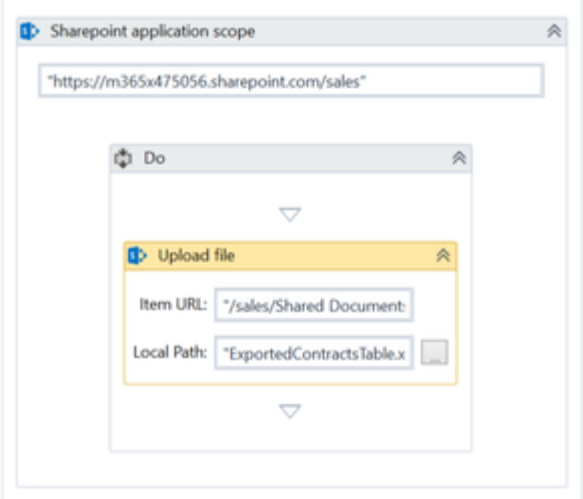


Let’s try to upload a file called ExportedContractsTable.xlsx to the test folder of the Documents Library. When we upload the document to this library, we would like to add values to both fields filled with values, so we will add the values to the **PropertiesToAdd** dictionary:

Properties Dictionary

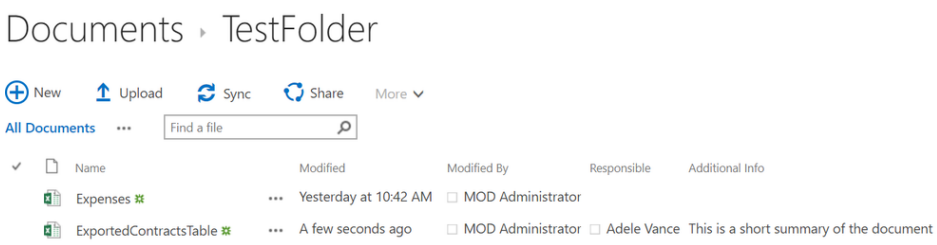
```
/* the FieldUserValue is initialized with the ID of the user*/
New Dictionary(Of String, Object) From {
{ "Additional_x0020_Info", "This is a short summary of the document"},
{ "Responsible", New Microsoft.SharePoint.Client.FieldUserValue() With {
.LookupId = 37 }}
}
```

The snippets that uploads the file will look something like this:



The Item URL is: **“/Shared Documents/TestFolder"**

The result will be:



3.6. Move Item

An activity that moves a file or folder from a specified URL in a Library, to another URL, either in the same Library or another one.

In the case of folders, they will be moved along with all their contents.

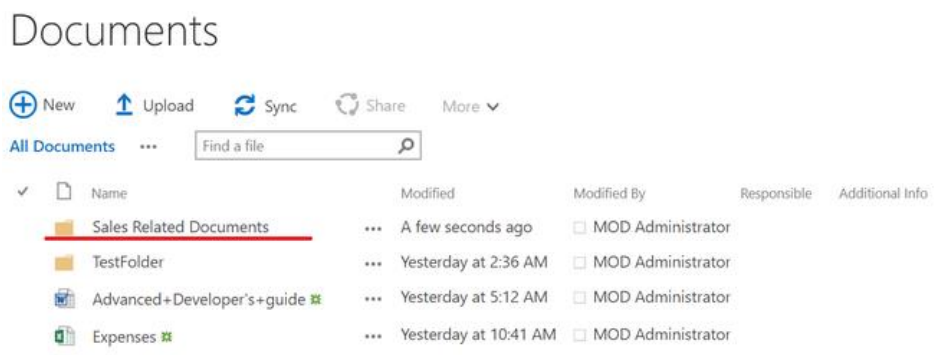
3.6.1. Parameters

- **RelativeUrl:** The relative URL of the file/folder that will be moved
- **DestinationRelativeUrl: The relative URL of the Folder/library** where the file/folder will be moved
- **AllowOverwrite:** Checkbox which specifies whether or not we should let the robot override a file with the same name already located at the Destination URL. Enabled by Default.

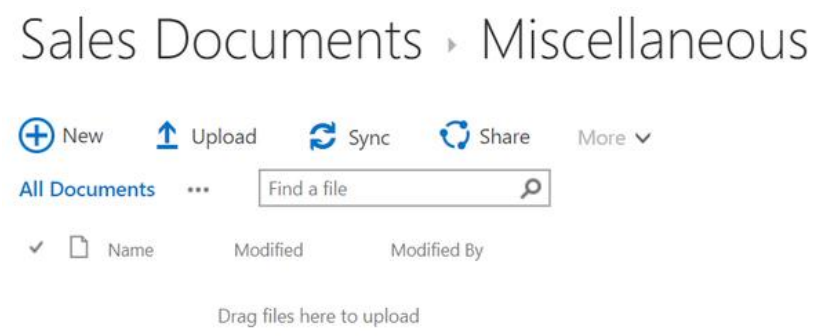
3.6.2. Example

Let's assume that inside my Sales site I want to move the **Sales Related Documents** folder from the **Documents** library to the Miscellaneous folder inside the **Sales Documents** library.

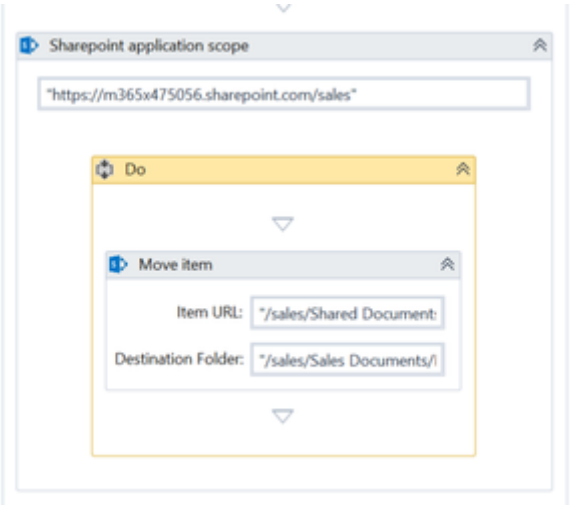
Folder to be moved:



Destination:

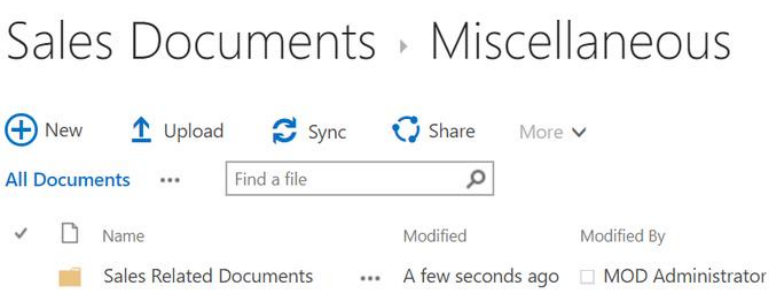


The snippet of code that achieves this:



The Source URL: **//Shared Documents/Sales Related Documents** and the Destination URL: **/Sales Documents/Miscellaneous.**

Result:



3.7. Rename Item

An activity that changes the name of a file of folder from a specified URL in a Library.

3.7.1. Parameters

- **RelativeUrl:** The relative URL of the file/folder that will be renamed.
- **NewName:** The new name of the file/folder

3.8. Check in File

An activity that checks in a file from a specified URL. If the file is not checked out, nothing happens.

3.8.1. Parameters

- **RelativeUrl:** The relative URL of the file

3.9. Check out File

An activity that checks out a file from a specified URL. If the file is already checked out, an exception will occur.

3.9.1. Parameters

- **RelativeUrl:** The relative URL of the file

3.10. Discard check out

An activity that discards the check out of a file from a specified URL. If the file is not checked out, an exception will occur.

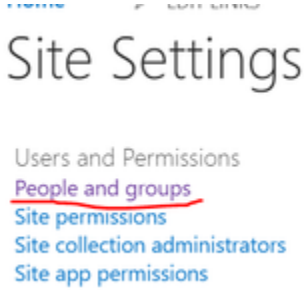
3.10.1. Parameters

- **RelativeUrl:** The relative URL of the file

4. SharePoint.Activities.Users

Groups are very important in SharePoint, they can be given permissions to read/edit/create/delete certain items, folders, lists libraries and even sites. They also have secondary purposes, serving as mailing lists, etc. SharePoint Groups can contain users and even AD groups, thus making managing permissions through groups very easy.

To check the current groups your Site has and their members, you should click the "Users and Groups" link inside the **Site Settings:**



This package contains a set of activities which can be used in order to create and delete user groups, as well as add and/or remove users from them:

- AddUsersToGroup
- CreateUserGroup
- RemoveGroup
- RemoveUserFromGroup

- GetUser
- GetAllUsersInsideGroup

All these activities can be used together with the **QueryGrouping** feature, except **GetAllUsersFromGroup** and **GetUserInformation**.

4.1. Create Group

This is an activity that creates a group on the site referenced by the SharePoint Application Scope.

4.1.1. Parameters

- **GroupName:** The name of the group we're about to create
- **GroupDescription:** A description for the group

4.2. Delete Group

Deletes one of the groups on your SharePoint Site.

4.2.1. Parameters

- **GroupName:** The name of the group we're about to delete

4.3. Add User to Group

Adds a user to a specific group. This also works with AD groups since in SharePoint they are pretty much treated just like users are.

4.3.1. Parameters

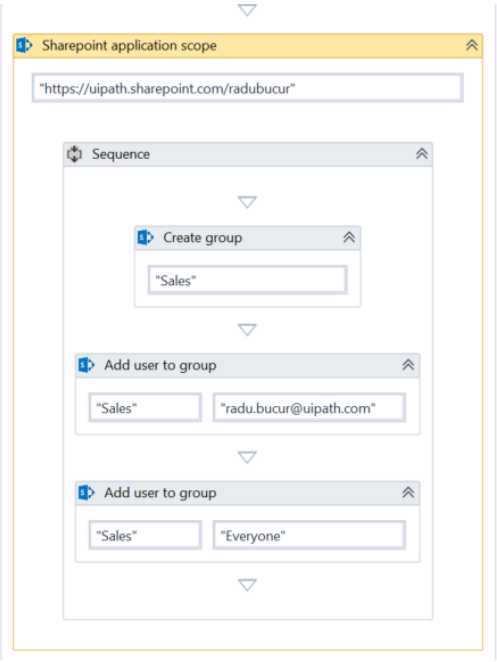
- **GroupName:** The name of the group we're adding the user to
- **User**: The users email or "{domain}\{username}" or the name of an AD Group

4.3.2. Example

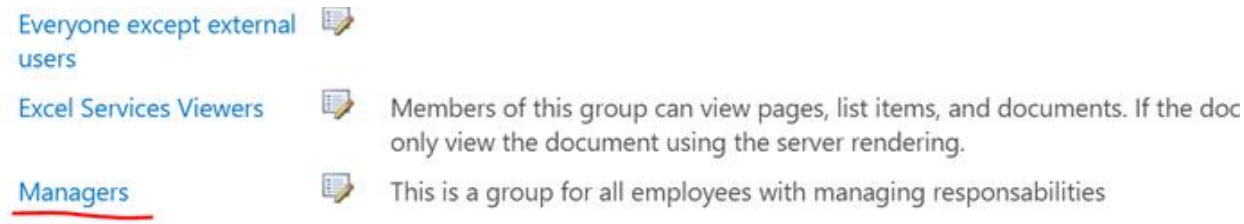
Let’s assume we want to create a group and add 2 users to it, a regular user, which we will reference using their email address and an AD Group called **Everyone.**

We will add the regular user to the group by referencing their email address and the AD group by using its name (which in this case is **Everyone**).

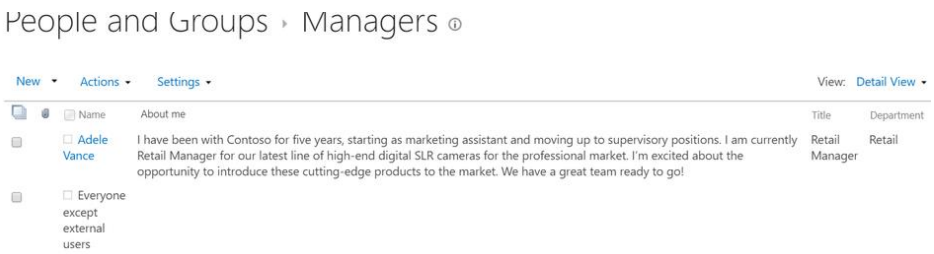
This example would look like this:



After the code above is executed on the groups page, we can find the following group:



Inside, we can find the following users:



4.4. Remove User From Group

Removes a user from a specific group. This also works for removing AD groups from inside SharePoint groups. It is very similar to the **Add User to Group** activity.

An exception will be thrown if this activity tries to remove a user that is not part of the group.

4.4.1. Parameters

- **GroupName:** The name of the group we're removing the user from
- **User**: The users email or "{domain}\{username}" or the name of an AD Group that needs to be removed from this group

4.5. Get User Information

This activity helps you search for a user in SharePoint and returns the users ID and full details. It searches the user by either the email or the display name of the user.

This activity returns an exception if more than one user is found or if the search has 0 results**. It is recommended that users are searched by email address, this is far more reliable as their First Name + Last Name combination might not be unique.**

4.5.1. Parameters

- **User:** The email/name the user will be searched by.

- **SharePointUser:** An **out** argument of type **Microsoft.SharePoint.Client.User** containing all the details of the user
- **UserID:** An **out** argument of type **int** containing the ID of the found user

4.6. Get All Users from Group

This activity allows you to get a list with all the users inside a SharePoint group. This is useful whenever we want to add/remove users to a group, since it can help us reduce the number of unnecessary queries done as well as preventing exceptions (since trying to remove a user which does not exist in a group can cause exceptions).

Using the GetAllUsersFromGroup activity will also give us all the relevant information about the users themselves.

This operation cannot be used if the QueryGrouping feature is enabled.

4.6.1. Parameters

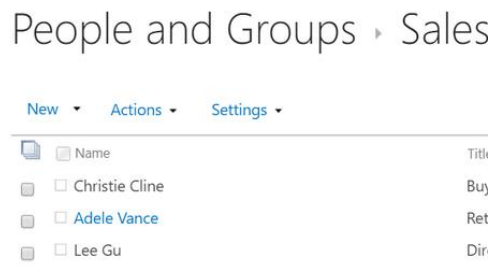
- **GroupName:** A string containing the name of the group we want to retrieve the users for

- **Result:** An **out** argument of type **List<Microsoft.SharePoint.Client.User>** containing all the users inside this group

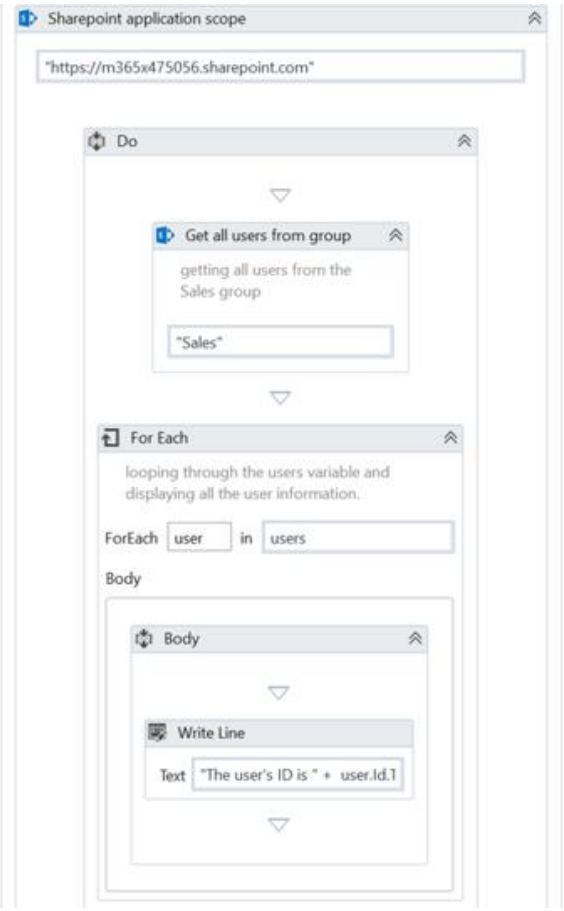
4.6.2. Example

Let's assume I want to display to the console all the members of the **Sales** group inside my Site Collection.

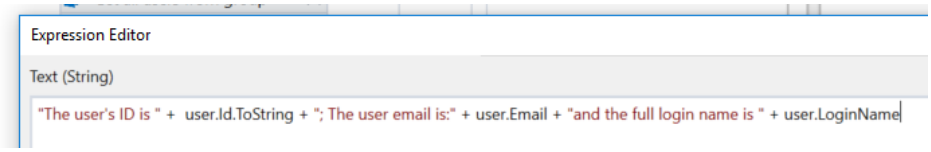
If I navigate to the group's page, I will see the following members:



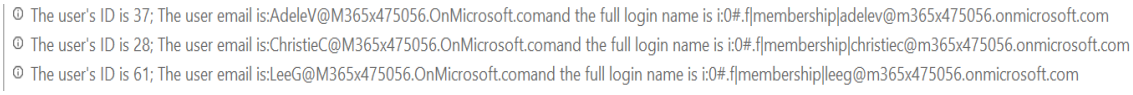
A workflow that retrieves the information of all these users from SharePoint and that displays it to the console, would look like this:



The Write line would retrieve the user information in the following manner:



The result inside the console would be:



5. SharePoint.Activities.Permissions

In order to provide access to our SharePoint content, we can assign permissions to groups or individual users at site, list/library, folder (or even item) level.

Since giving permissions to users directly is a bad practice, we only allowed the assignation and removal of permissions to and from groups only.

An alternative to it would be to simply create a group which contains the users who should receive the permission and assign that permission directly to this group and then add the users to it.

With our activities we can use all the permission levels SharePoint has:

- View Only
- Limited Access
- Read
- Contribute
- Edit
- Design
- Full Control

The following activities allow adding and removing permissions for:

- the SharePoint site itself
- a list
- a library
- or any folder inside a list/library

This is a set of activities which can be used in order to add and/or remove the permissions a group has **in relation to a list or library or the current site**:

- AddPermission

- RemovePermission
- GetAllPermissionsFromGroup

5.1. Add Permission

Adds a permission to specific group. This activity is compatible with the **QueryGrouping** option.

5.1.1. Parameters

- **GroupName:** the name of the group we want to assign permissions to.
- **PermissionToGive**: a dropdown that allows the user to select the permission level.
- **ListName:** the name of the list/library we want to assign permissions to. **If this is left empty, the permissions will be assigned to the SharePoint Site instead**
- **ListType:** this parameter needs to be used only when interacting with the permissions of a folder inside either a list or a library. It should be set to "Library" or "List" depending on the type of SharePoint collection we want to interact with.
- **FolderPath:** the folder inside the list/library we want to assign permissions to. **If this is left empty, the permissions will be assigned directly to the list/library.**
    - This property can be used only if the ListName is not null

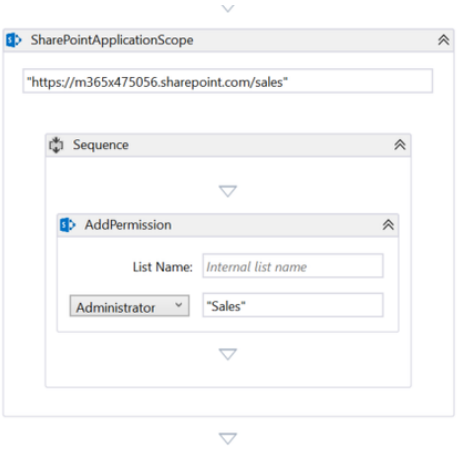
Observation:

SharePoint Online does not allow altering the permissions of the root site of a site collection, only the sub-sites can have their permissions changed. **Trying to either remove or add permissions to the root site of a site collection will result in an exception!**

5.1.2. Example

Let's assume that I have a sub-site in my SharePoint collection dedicated to the Sales team, a group named **Sales** which contains all the Sales Department employees and that I need to give them full control on that site.

I can do that very easily using the following workflow:



Notice that the List parameter is empty since the permissions are applied directly to the site. After I open the site's permission page, I will notice the following row:

5.2. Remove Permission

Remove all permissions a group has in relation to another list/library, its folder or even the SharePoint site.

This activity is compatible with the **QueryGrouping** option enabled.

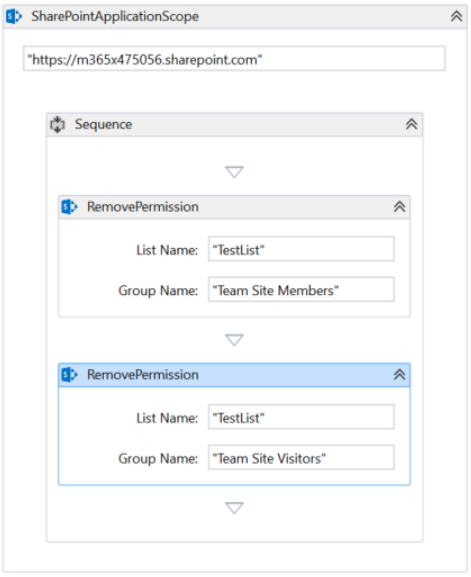
5.2.1. Parameters

- **GroupName:** the name of the group we want to remove permissions for.
- **ListName:** the name of the list/library we want to delete permissions from. **If this is left empty, the permissions will be removed from the SharePoint Site instead**
- **ListType:** this parameter needs to be used only when interacting with the permissions of a folder inside either a list or a library. It should be set to "Library" or "List" depending on the type of SharePoint collection we want to interact with.
- **FolderPath:** the folder inside the list/library we want to delete permissions from. **If this is left empty, the permissions will be removed directly from the list/library.**
    - This property can be used only if the ListName is not null

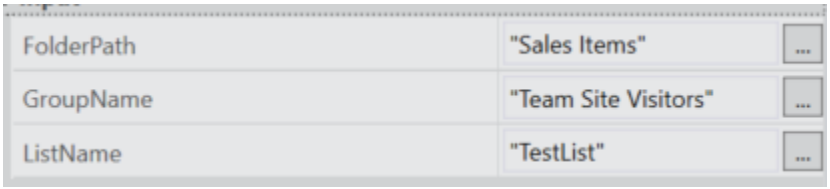
5.2.2. Example

Let's assume that I have a list called **TestList** where I have a folder **Sales Items** that we do not want to be accessible/visible to the users in the groups **Team Site Users** and **Teams Site Visitors**. We can achieve that by simply removing the permissions these 2 groups have on that folder.

The workflow that does this would look something like this:



Note that both the Remove Permission activities reference the Sales Items folder in their **FolderPath** argument:



5.3. Get All Permissions

Gets all existing permissions a list/library, folder or even SharePoint Site has. This is very helpful if the user is trying to either add or remove any permissions in a SharePoint site, since it can help us reduce the number of unnecessary queries done as well as preventing exceptions (since trying to remove a permission which does not exist for a group can cause exceptions).

This activity is not compatible with the **QueryGrouping** option enabled.

5.3.1. Parameters

- **ListName:** the name of the list/library we want to delete permissions from. **If this is left empty, the permissions will be removed from the SharePoint Site instead**
- **FolderPath:** the folder inside the list/library we want to delete permissions from. **If this is left empty, the permissions will be removed directly from the list/library.**
    - This property can be used only if the ListName is not null

- **ListType:** this parameter needs to be used only when interacting with the permissions of a folder inside either a list or a library. It should be set to "Library" or "List" depending on the type of SharePoint collection we want to interact with.

- **Result:** an **out** argument of type List<Tuple<string,string>> which contains a full list with all the permissions on the specified SharePoint object
    - **if the entity that has this permission is a user** then the first value in each tuple **contains the full login name of the user (The full login name is a bit ugly but usually has an easily identifiable format!)**
    - **if the entity that has this permission is a group** then the first value in each tuple **contains the group's name**
    - the second value of each tuple contains the permissions name

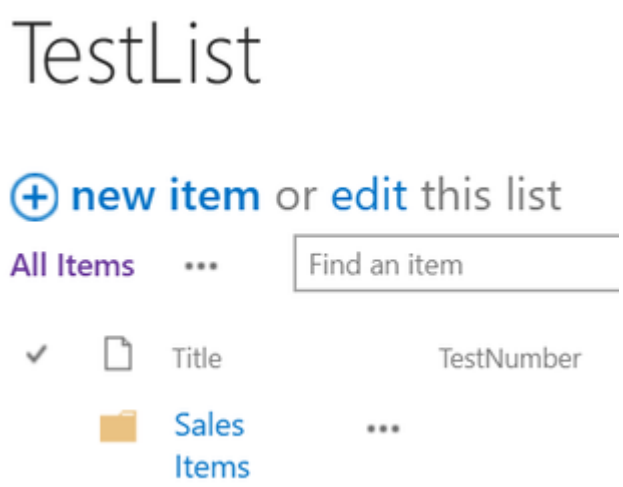
Observation:

If an entity has multiple permissions, we will have a tuple for each one of them in the output list!

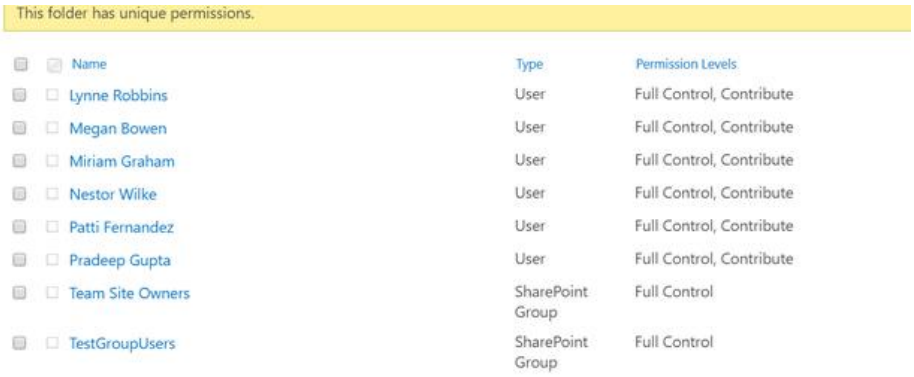
5.3.2. Example

Let's assume I want to display all the permissions my users have on the **Sales Items** folder in my **TestList.**

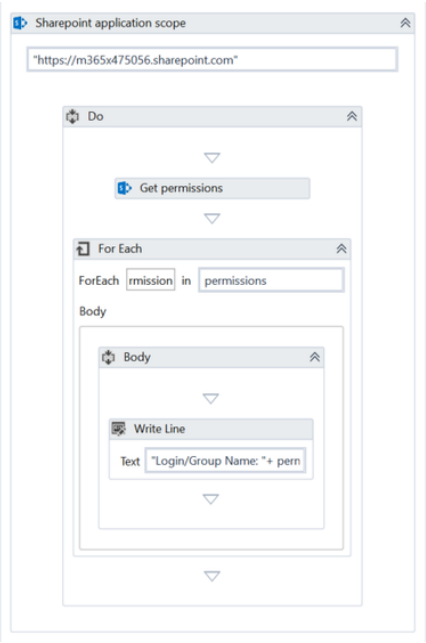
Below we can see the folder:



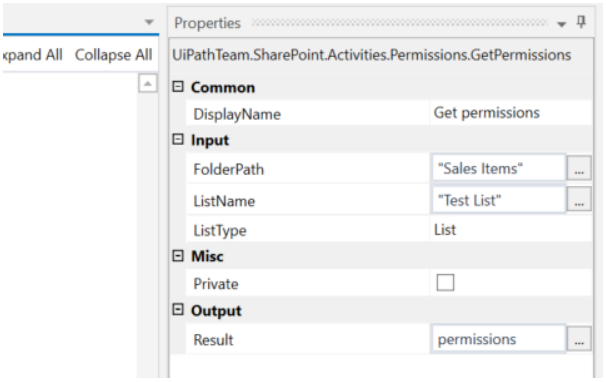
If I navigate to folders permission page, I can see there are plenty of permissions assigned to it (some inherited from the parent list and some assigned specifically to it, it doesn't matter, they will all be retrieved):



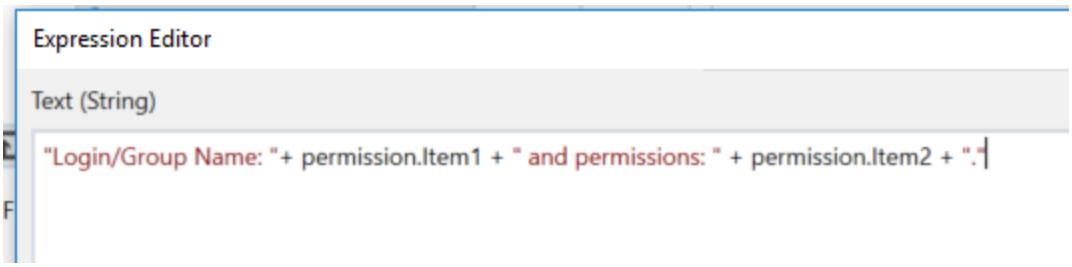
Using the following workflow, I can Retrieve and display to the console all these permissions:



The arguments my Get Permissions activity will have are the following:



The Write Line will access the properties in each tuple the following way:



The console output will be:

```
Login/Group Name: Team Site Owners and permissions: Full Control.
Login/Group Name: i:0#.f|membership|meganb@m365x475056.onmicrosoft.com and permissions: Full Control.
Login/Group Name: i:0#.f|membership|meganb@m365x475056.onmicrosoft.com and permissions: Contribute.
Login/Group Name: i:0#.f|membership|pradeepg@m365x475056.onmicrosoft.com and permissions: Full Control.
Login/Group Name: i:0#.f|membership|pradeepg@m365x475056.onmicrosoft.com and permissions: Contribute.
Login/Group Name: i:0#.f|membership|nestorw@m365x475056.onmicrosoft.com and permissions: Full Control.
Login/Group Name: i:0#.f|membership|nestorw@m365x475056.onmicrosoft.com and permissions: Contribute.
Login/Group Name: i:0#.f|membership|lynner@m365x475056.onmicrosoft.com and permissions: Full Control.
Login/Group Name: i:0#.f|membership|lynner@m365x475056.onmicrosoft.com and permissions: Contribute.
Login/Group Name: i:0#.f|membership|miriamg@m365x475056.onmicrosoft.com and permissions: Full Control.
Login/Group Name: i:0#.f|membership|miriamg@m365x475056.onmicrosoft.com and permissions: Contribute.
Login/Group Name: i:0#.f|membership|pattif@m365x475056.onmicrosoft.com and permissions: Full Control.
Login/Group Name: i:0#.f|membership|pattif@m365x475056.onmicrosoft.com and permissions: Contribute.
Login/Group Name: Limited Access System Group and permissions: Limited Access.
Login/Group Name: SharePointHome OrgLinks Admins and permissions: Limited Access.
Login/Group Name: SharePointHome OrgLinks Editors and permissions: Limited Access.
Login/Group Name: SharePointHome OrgLinks Viewers and permissions: Limited Access.
Login/Group Name: TestGroupUsers and permissions: Full Control.
Login/Group Name: TestGroupUsers and permissions: Limited Access.
```

Query Grouping

The **QueryGrouping** option is a feature which allows the users to group up repetitive queries in batches, so that they are processed more efficiently. Instead of simply sending each query individually to the server and waiting for the response to be received, the scope activity just stores all the queries and sends them after all the children activities got executed resulting in a more efficient processing time.

However, this is only possible for the following activities: **AddListItem, AddPermission, RemovePermission, AddUserToGroup, CreateGroup, DeleteGroup, RemoveUserFromGroup.** For the rest of the activities it would either not make sense (it does not make sense to use the **ReadListItems** activity in an async way since it would not be able to return the items, or the **EditListItems** activity since it only uses 1 query to update multiple items) or it is not possible since some activities require multiple queries executed one after another in order to perform one action.

As it was mentioned before, the QueryGrouping feature can be enabled by clicking the **QueryGrouping** Checkbox in the SharePoint Scope Activity:

Observation:

Depending on the size of the data we might try to send to the server at once, the request might receive the following exception: **"The request message is too big. The server does not allow messages larger than XXXXX bytes."**

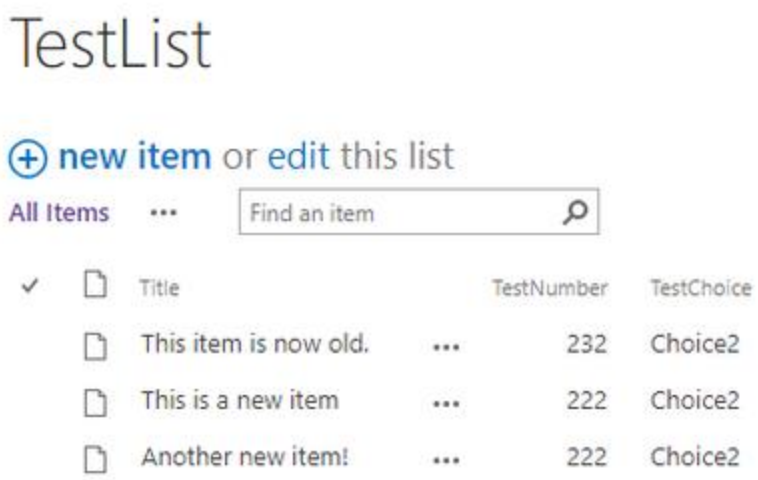
In order to avoid that, for queries that send a large amount of data to the server, the number of queries inside a SharePoint Scope should be limited (a good example can be seen below).

If this feature is enabled and the user tries to add an activity that is not supported, a validation message will appear.

Example

Let's prepare a small example in which we need to add all the rows from an excel file to our SharePoint list (called **TestList**), assuming that our file has hundreds of rows, adding them one by one would take a lot of time so in order to speed up the creation of the list items, we will try to add them 100 at a time.

The list we'll add everything on:



The contents of the excel file (in total 211 rows):

C7 123

	A	B	C	D
1	Title	TestChoice	TestNumber	
2	New Item 1	Choice3	123	
3	New Item 2	Choice2	123	
4	New Item 3	Choice3	123	
5	New Item 4	Choice1	123	
6	New Item 5	Choice1	123	
7	New Item 6	Choice2	123	
8	New Item 7	Choice3	123	
9	New Item 8	Choice1	123	
10	New Item 9	Choice3	123	
11	New Item 10	Choice2	123	
12	New Item 11	Choice3	123	
13	New Item 12	Choice1	123	
14	New Item 13	Choice1	123	

The full example with explanatory annotations can be found here (SharePoint Login Data removed, of course).

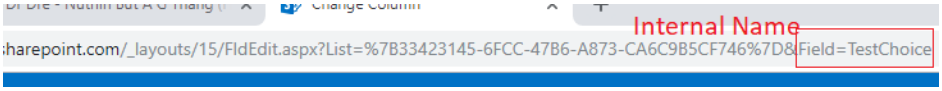
SharePoinQueryGroupingTest.zip

Prerequisites

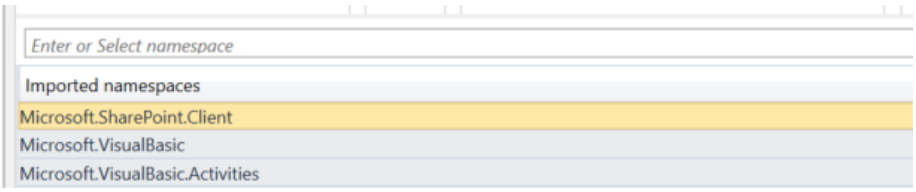
- Have access to an instance of SharePoint and an account with all the necessary permissions. **You will not be able to use this package using your credentials to do any operations that you couldn't do on your SharePoint environment in the browser.**
- This solution might not work if your SharePoint instance is using a 3rd party Identity Provider.

Observations

- Make sure that when you are working on a SharePoint site **you use the correct URL** in the SharePoint scope and not the URL of a parent site and/or of a sub-site.
- Keep in mind that the **QueryGrouping** option is only available for some activities (mentioned previously). For any activity that has Out Arguments and it is used with the query grouping feature enabled, it will have those arguments return **null.**
- For all List Activities using list fields, we must reference these fields using their **Internal Name,** not their title (often enough, they are not the same). In order to obtain the internal name of a field open the list settings, click the field and look at the URL of the page, the internal name will be there:



- If you want to assign a value to field specific field for a list item, **first make sure that field exists inside your list** (otherwise create it).
- Since giving permissions to users directly is a bad practice, **we only allowed the assignation and removal of permissions to and from groups only. This package does not allow the assignation and/or removal of permissions from users directly!**
- **Each time you alter the permissions of an object, you will break the inheritance of permissions from its parent element. This means that if the permissions of the parent are changed, the changes will not be reflected on the original element.** Try to be careful of the scope of the permissions you assign and always assign permissions to the highest suitable scope.
- Some of the activities (GetAllUsersFromGroup, GetUser or some of the list items using more advanced data fields) can use types and classes that are specific to the **Microsoft.SharePoint.Client** if you are having issues with these types, please make sure that this namespace is added inside the imports Tab:



- We provided plenty of detailed examples so if you're having any issues using this package, make sure to consult the examples.
- If the metadata needs to be edited for Files, the **Update List Items** Activity can be used on the parent library!

Technical Approach

This package is mainly built using the .NET **Client Side Object Model** (CSOM) which contains a large number of object representing SharePoint objects which can be used in order to make changes and retrieve information from the SharePoint site. Since CSOM is very similar for all different types of SharePoint instances, we can use the same set of activities for both SharePoint Online and SharePoint OnPremises.

Another advantage is that we can choose the moment we send the queries created so far to the server, so in some cases we can group them up and send them together to the server, therefore increasing the efficiency of the package.

Additionally, CSOM can leverage the Collaborative Application Markup Language (CAML) which is a very powerful XML-based language that can be used to create extremely detailed and

complex queries on SharePoint Lists. These queries greatly increase the versatility of the activities and the big advantage is that several 3rd party tools can be used to generate CAML queries without the user having any previous knowledge of this language.

In some places, the REST API of the SharePoint instance is used in order to download and/or upload documents in order to avoid issues regarding the size of documents.

Below you can find more info regarding:

1. CSOM and SharePoint REST API: https://docs.microsoft.com/en-us/sharepoint/dev/sp-add-ins/sharepoint-net-server-csom-jsom-and-rest-api-index
2. CAML: https://docs.microsoft.com/en-us/sharepoint/dev/schema/collaborative-application-markup-language-caml-schemas
3. SmartCAML (3rd party tool that can generate CAML query syntax): https://www.microsoft.com/en-us/p/smartcaml/9nn8gjpnxvfg