Introduction

Updating the data column values of a data table is a common task in many automation processes. This blog will present the different available options to find the best fitting approach for your own scenario.

Use cases step by step

1. **1. Case: update column values on all data rows**

In this section, different approaches are presented, while updating column values on all data rows of a data table.

1. **1.1 For Each Row / For Each Row in Data Table approach**

With the help of this activity, a data table can be looped over all its data rows, and the particular data columns can be set or updated.

1. 1.1.1 Sample Input and Assignment:

**Input:**

|  |  |  |
| --- | --- | --- |
| 1. PRODUCTID (String) | NETPRICE(Double) | GROSSPRICE |
| 0001 | 100 |  |
| 0002 | 250 |  |
| 0003 | 300 |  |
| 0004 | 450 |  |

**Assignment:**

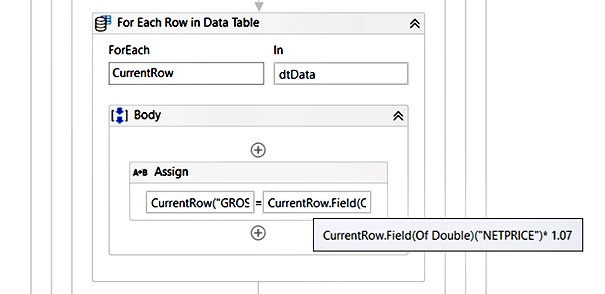
The GROSSPRICE data column value is to be updated by the NETPRICE value multiplied by 1.07. This will represent that 7% of the NETPRICE is to be added to the GROSSPRICE.

Result:

|  |  |  |
| --- | --- | --- |
| PRODUCTID (String) | NETPRICE(Double) | GROSSPRICE |
| 0001 | 100 | 107 |
| 0002 | 250 | 267.5 |
| 0003 | 300 | 321 |
| 0004 | 450 | 481.5 |

1.1.2 Implementation

The activity will loop over all data rows where the current looped data row is referenced with CurrentRow. Setting the calculated GROSSPRICE value can be done within the shortened statement within an assign activity:



***General Syntax:***

**CurrentRow(ColumnNameOrIndex) = CurrentRow.Field(Of Double)(ColumnNameOrIndex)\* 1.07**

***Examples***

**CurrentRow("GROSSPRICE") = CurrentRow.Field(Of Double)("NETPRICE")\* 1.07**

**CurrentRow(2) = CurrentRow.Field(Of Double)(1)\* 1.07**

**Kindly note:** the column index is zero based

1. **1.2**[**Invoke Code**](https://docs.uipath.com/activities/docs/invoke-code)**activity approach**

The Invoke Code activity can be used to execute VB.Net or C# Code.

1.2.1 Sample Input and Assignment:

|  |  |  |
| --- | --- | --- |
| Input |  |  |
| PRODUCTID | NETPRICE | GROSSPRICE |
| 0001 | 100 |  |
| 0002 | 250 |  |
| 0003 | 300 |  |
| 0004 | 450 |  |

|  |  |  |
| --- | --- | --- |
| Output |  |  |
| PRODUCTID | NETPRICE | GROSSPRICE |
| 0001 | 100 | 107 |
| 0002 | 250 | 267.5 |
| 0003 | 300 | 321 |
| 0004 | 450 | 481.5 |

**Assignment**: update: GROSSPRICE by NETPRICE \* 1.07 (add 7%)

1. 1.2.2 Implementation

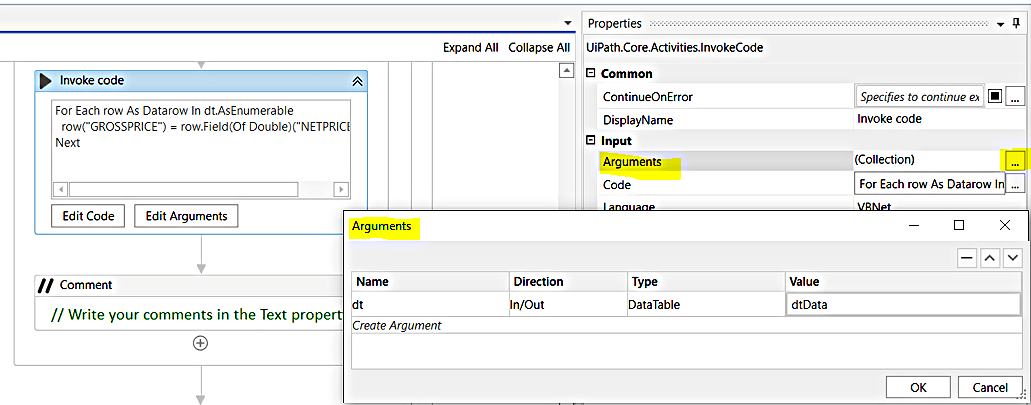
The example from above, done with a for each row / for each row in data table activity is mirrored to the essential building blocks offered from .Net (VB.Net in this case). It loops over all data rows and updates the GROSSPRICE column value.

**Code**:

For Each row As Datarow In dt.AsEnumerable

 row("GROSSPRICE") = row.Field(Of Double)("NETPRICE")\* 1.07

Next



**1.3** **Data Column Expression Approach**

With an expression defined for a particular data column its value is computed and is retrievable.

1. 1.3.1 Sample Input & Assignment:

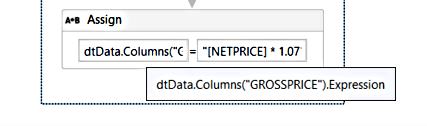
|  |  |  |
| --- | --- | --- |
| PRODUCTID | NETPRICE | GROSSPRICE |
| 0001 | 100 |  |
| 0002 | 250 |  |
| 0003 | 300 |  |
| 0004 | 450 |  |

|  |  |  |
| --- | --- | --- |
| PRODUCTID | NETPRICE | GROSSPRICE |
| 0001 | 100 | 107 |
| 0002 | 250 | 267.5 |
| 0003 | 300 | 321 |
| 0004 | 450 | 481.5 |

Assignment: Update: GROSSPRICE by NETPRICE \* 1.07 (add 7%)

1. 1.3.2 Implementation

The data column class offers a property with the name Expression. By assigning the computation expression on this property, the value of the data column will be computed accordingly.



*General Syntax:*   
yourDataTableVar.Columns(ColumnNameOrIndex).Expression = “TheExpressionString”

*Examples*   
dtData("GROSSPRICE").Expression = "[NETPRICE]\* 1.07”   
CurrentRow(2).Expression = "[NETPRICE]\* 1.07"

Kindly note: the column index is zero based



**1.4 Language Integrated Query** **(LINQ)** **approach**

**LINQ** is a part of the .NET framework and offers capabilities for interacting with data, objects, or other sources within a particular syntax.

1. 1.4.1 Sample Input & Assignment

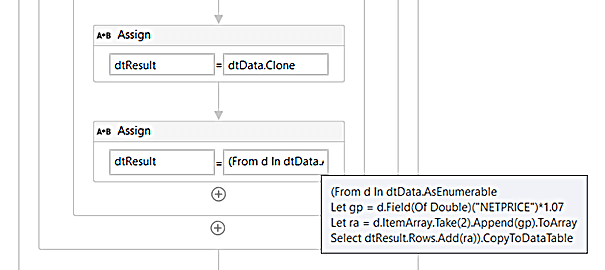
|  |  |  |
| --- | --- | --- |
| PRODUCTID | NETPRICE | GROSSPRICE |
| 0001 | 100 |  |
| 0002 | 250 |  |
| 0003 | 300 |  |
| 0004 | 450 |  |

|  |  |  |
| --- | --- | --- |
| PRODUCTID | NETPRICE | GROSSPRICE |
| 0001 | 100 | 107 |
| 0002 | 250 | 267.5 |
| 0003 | 300 | 321 |
| 0004 | 450 | 481.5 |

Assignment: Update: GROSSPRICE by NETPRICE \* 1.07 (add 7%)

1. 1.4.2 Implementation

There are several techniques of using LINQ for processing the data table. The used approach in this implementation is about data table reconstruction by populating an empty cloned data table with existing and newly calculated data column values.



**Step 1:**Preparation

A data table variable with the name dtResult will be cloned from an existing data table (dtData). Initially dtResult is empty but will have the same data column structure as dtData.

**Step 2:** Executing the LINQ Statement and assignment of the result to dtResult

**Explanation: LINQ statement**

|  |
| --- |
| (From d In dtData.AsEnumerable |
| Similar to the For Each Row Activity, d will reference the current looped data row |
| Let gp = d.Field(Of Double)("NETPRICE")\*1.07 |
| Calculate the GROSSPRICE and memorize it for later, referenced by gp |
| Let ra = d.ItemArray.Take(2).Append(gp).ToArray |
| Construct the data row ItemArray by using the first two column values and append the calculated gp on the end – referenced by ra e.g. {"0003", 300, 321} |
| Select dtResult.Rows.Add(ra)).CopyToDataTable |
| Add a data row to dtResult by using the rowArray (ra) |

2. **2. Case: update column values on specific data rows**

The recommended strategy for updating the data column value on specific data rows is to reduce in advance the set of data rows to the required subset of data rows. This section uses some options for filtering data tables, but can also be implemented with other filtering data table approaches. There are several approaches available in [UiPath Studio](https://www.uipath.com/product/studio). For more details on filtering data tables, refer to the [UiPath Community blog](https://www.uipath.com/community/rpa-community-blog): *“*[*How to Filter Data Tables in UiPath Studio*](https://www.uipath.com/community/rpa-community-blog/four-ways-to-filter-data-table-in-uipath-studio)*”.*

1. 2.1 For Each Activity Approach with LINQ filtering

The result of filtered data rows is passed to the For Each Activity. Column values will be updated within For Each Activity block.

1. 2.1.1 Sample Input & Assignment

|  |  |  |
| --- | --- | --- |
| Team | Status | Comment |
| 0001A | moved | Internal |
| 0001B | moved | internal |
| 0002A | *scheduled* | external |
| 0002B | *scheduled* | internal |
| 0003A | unplanned |  |
| 0003B | unplanned |  |

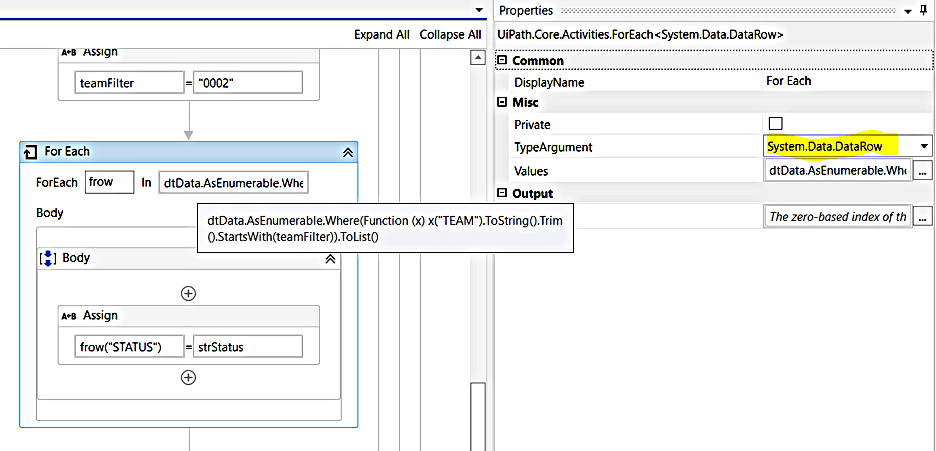
Let's update all data rows where the team id (column TEAM) starts with 0002. Set the status to “moved”.

Result:

|  |  |  |
| --- | --- | --- |
| Team | Status | Comment |
| 0001A | moved | Internal |
| 0001B | moved | internal |
| 0002A | moved | external |
| 0002B | moved | internal |
| 0003A | unplanned |  |
| 0003B | unplanned |  |

1. 2.1.2 Implementation

With the help of LINQ, the looped rows can be reduced to the only relevant ones, which are to be updated. Within the For Each Activity, the data column value of resulting data rows will be updated.



In preparation, the filter value for a given team set is assigned to a variable (teamFilter). With an LINQ statement, only the relevant data rows are pre-filtered. The data rows to be updated are changed to the new status value from the strStatus variable

LINQ for the filtering:

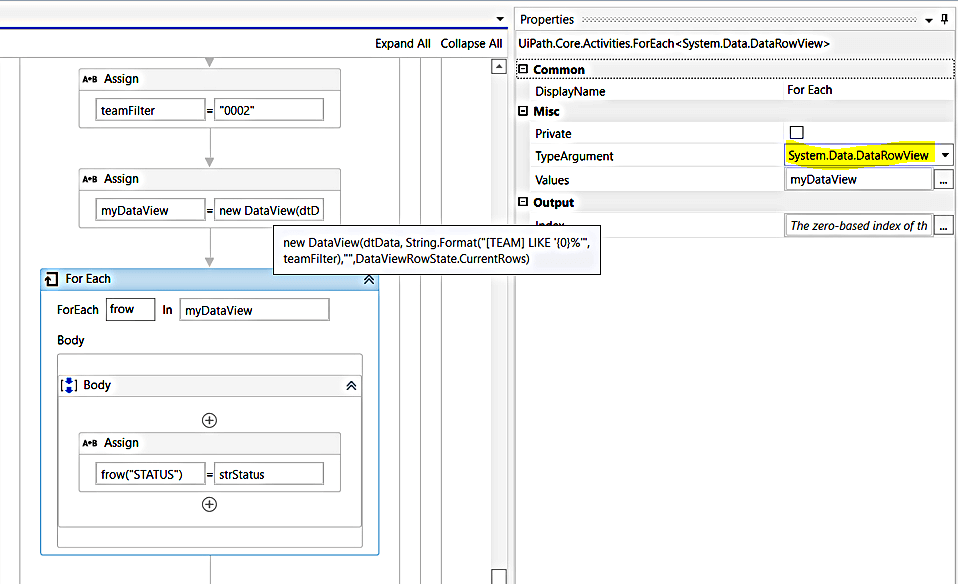
 dtData.AsEnumerable.Where(Function (x) x("TEAM").ToString().Trim().StartsWith(teamFilter)).ToList

1. 2.2 Other Alternative Approaches

In alternative approaches, the part of updating the data column values and/or filtering the data rows can be modified with different options. Updating the data column values can be done for example within an Invoke Code Activity as presented above. And filtering the data rows can be done with other available approaches e.g., using the Datatable class Select method:

YourDataTableVar.Select(<DataRow Filter Expression String>)

**Example**: using the For Each Activity in combination with a DataView of the data table:



On the top of a data table, the data view will be defined. The data view is configured to offer only the data rows where the value from the TEAM column starts with the value from the teamFilter variable: 0002. From all data rows of the underlying data table, the data view will filter out the rows as defined within the filter expression and will pass it to the For Each Activity. Within the For Each block the STATUS column value will be set to the value of the variable strStatus.

With data view a dynamic subset of data rows is provided, without removing the other data rows from the underlying data table. Fulfilling certain constraints, the filtered data rows and their column values can be updated, and the updates are done on the underlying data table.

[*Review UiPath*](https://eur02.safelinks.protection.outlook.com/?url=https%3A%2F%2Fgtnr.io%2FpixDNAqLA&data=04%7C01%7Coana.chicioroaga%40uipath.com%7C1597a34c2954483db1b108d9a50e0b0a%7Cd8353d2ab1534d178827902c51f72357%7C0%7C0%7C637722302001252380%7CUnknown%7CTWFpbGZsb3d8eyJWIjoiMC4wLjAwMDAiLCJQIjoiV2luMzIiLCJBTiI6Ik1haWwiLCJXVCI6Mn0%3D%7C1000&sdata=hSUSDJkAq06f1yUNHHXGGJPY0c2jcNKk%2B5hy4NzedCg%3D&reserved=0)*and earn gift cards. Join us and help other professionals learn from your experience with UiPath. Your review can be anonymous.*

1. Conclusion

When updating the column values of a data table, the activities like **for each / for each row / for each row in data table** are very useful and a first choice. This approach allows easy debugging of the code in case of any failure for further analysis.

The approach of **Data Column Expression** realizes the provisioning of the data column values by a rule applied on the entire data table for all data rows. This has the benefit that the computed value is also available when new data rows are added, in contrast to the **for each X**activity approach or **LINQ-based** implementations. But the Data Column Expression Approach relies strictly on the data column’s data type. There can be scenarios where this constraint is less controllable, e.g., when manual entries in Excel are processed.

**LINQ based approaches** can exploit the power of LINQ when targeting more compact implementations. It's recommended to align the parts within the LINQ statement, to provide good support in order to check the processing, e.g., in extended debugging. When using a LINQ-based approach with a reconstruction of the data table, it should be considered that the data is additionally held in the memory. Therefore, it should always be checked whether an alternative implementation is more adequate and straightforward for the current scenario.

When only a **subset of data rows** and its data column values are needed to update, then building the subset of the relevant data rows is required. Building this subset of data rows is usually done with different options for **filtering a data table** available in [UiPath Studio](https://www.uipath.com/product/studio) (check out the tutorial in the [UiPath Community blog:](https://www.uipath.com/community/rpa-community-blog)*“*[*How to Filter Data Tables in UiPath Studio*](https://www.uipath.com/community/rpa-community-blog/four-ways-to-filter-data-table-in-uipath-studio)*”)*. Once the subset of relevant data table rows is built, the filtered data rows will be processed, and its column values are updated with one of the corresponding approaches presented in this blog.

The [**Invoke Code Activity Approach**](https://docs.uipath.com/activities/docs/invoke-code) can be used when a programming-oriented implementation is focused. Similarly, the pattern of For Each X Activity and conditional Activities like If, Switch, etc. can be used for custom implementations for the data column value update part and optionally for the data row filtering part. Such an implementation practice can risk the re-implementation of already existing activities or approaches. It is recommended to check if this type of re-implementation can be avoided by using already available options instead.