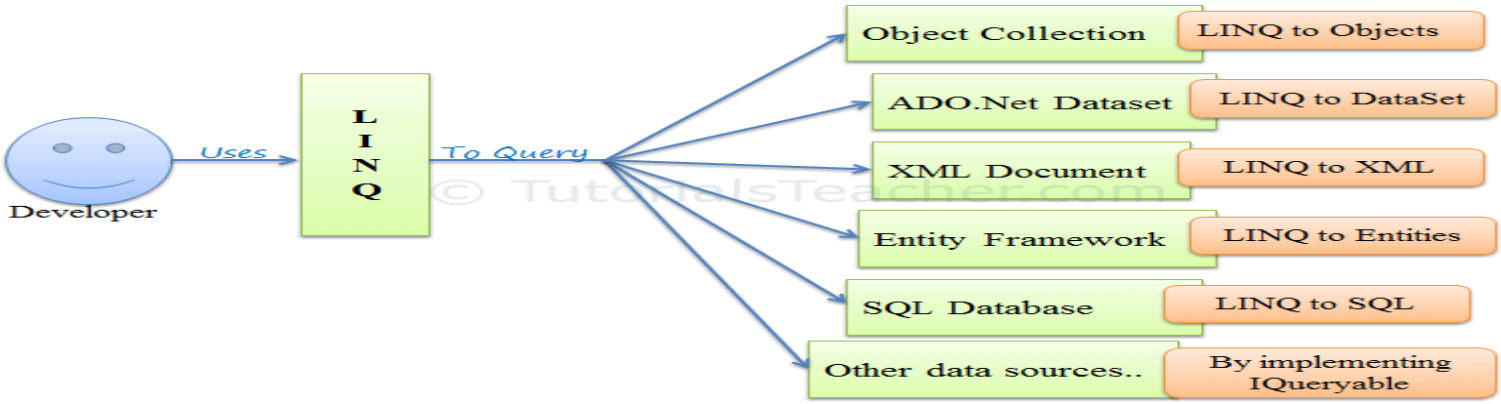
LINQ IN UIPATH

What is Linq in UiPath?

LINQ stands **for Language-Integrated Query** and enables you to query data in a simple and powerful way. In UiPath, LINQ is commonly used to query arrays, data tables, dictionaries, lists, and XML. Arrays, dictionaries, and lists require no modifications or special treatment to act as a LINQ data source

LINQ (Language Integrated Query) is uniform query syntax in C# and VB.NET to retrieve data from different sources and formats. It is integrated in C# or VB, thereby eliminating the mismatch between programming languages and databases, as well as providing a single querying interface for different types of data sources.

For example, SQL is a Structured Query Language used to save and retrieve data from a database. In the same way, LINQ is a structured query syntax built in C# and VB.NET to retrieve data from different types of data sources such as collections, ADO.Net DataSet, XML Docs, web service and MS SQL Server and other databases.



LINQ Usage

LINQ queries return results as objects. It enables you to uses object-oriented approach on the result set and not to worry about transforming different formats of results into objects.

LINQ query operators(Select, where, any, all etc) are implemented in enumerable class as extenstion methods on the IEnumerable <t> interface

What is Extension method?

Adding additional methods to an existing type or class without modifying the source code

Why we need LINQ?

LINQ is a way that **helps you to transform ANY source of data in ANY format that you want**: from Excel files to files and UiPath Selectors. By using LINQ to develop robots, you will: Decrease the number of activities. ... Make your robot work faster.

Primary feature of Linq query that I feel is the power of LINQ query is the time saving like when we are dealing with activities especially(Uipath Activities) what we typically tend to do is if you want to perform an action on each item of array or list what we tends to do is we simply perorm an action on each item one by one using For Loop and then append the data in some other collection

Advantage of LINQ

- **Familiar language:** Developers don’t have to learn a new query language for each type of data source or data format.
- **Less coding:** It reduces the amount of code to be written as compared with a more traditional approach.
- **Readable code:** LINQ makes the code more readable so other developers can easily understand and maintain it.
- **Standardized way of querying multiple data sources:** The same LINQ syntax can be used to query multiple data sources.
- **Compile time safety of queries:** It provides type checking of objects at compile time.
- **IntelliSense Support:** LINQ provides IntelliSense for generic collections.
- **Shaping data:** You can retrieve data in different shapes.
- LINQ also allows debugging which can be useful while troubleshooting.
- Most of the LINQ queries are reusable.
- LINQ provides powerful filtering, ordering, and grouping capabilities with minimum application code.

Linq query execution

They are two Types

1 Deffered Execution

2 Immediate Execution

Deffered Execution

Query Executes when you iterate the element and not when you create the query

Immediate Execution

Query Executes immediately when you convert your query to an in-memory object

LINQ OPERATORS

Linq operators are divided into following categories based on their functionality

Aggregation

Conversion

Element

Generation Operations

Grouping Operations

Joining

Ordering

Partitioning

Quantifiers

Restriction

Join Operator

Joining refers to an operation in which data sources with difficult to follow relationships with each other in a direct way are targeted.

Operator	Description	C# Query Expression Syntax	VB Query Expression Syntax
Join	The operator join two sequences on basis of matching keys	join … in … on … equals …	From x In …, y In … Where x.a = y.a
GroupJoin	Join two sequences and group the matching elements	join … in … on … equals … into …	Group Join … In … On …

Filtering Operators

Filtering is an operation to restrict the result set such that it has only selected elements satisfying a particular condition

Operator	Description	C# Query Expression Syntax	VB Query Expression Syntax
where	Filter values based on a predicate function	where	Where
OfType	Filter values based on their ability to be as a specified type	Not Applicable	Not Applicable

Grouping Operators

The operators put data into some groups based on a common shared attribute.

Show Examples

Operator	Description	C# Query Expression Syntax	VB Query Expression Syntax
GroupBy	Organize a sequence of items in groups and return them as an IEnumerable collection of type IGrouping<key, element>	group … by -or- group … by … into …	Group … By … Into …
ToLookup	Execute a grouping operation in which a sequence of key pairs are returned	Not Applicable	Not Applic

Aggregation

Performs any type of desired aggregation and allows creating custom aggregations in LINQ

Operator	Description	C# Query Expression Syntax	VB Query Expression Syntax
Aggregate	Operates on the values of a collection to perform custom aggregation operation	Not Applicable	Not Applicable
Average	Average value of a collection of values is calculated	Not Applicable	Aggregate … In … Into Average()
Count	Counts the elements satisfying a predicate function within collection	Not Applicable	Aggregate … In … Into Count()
LonCount	Counts the elements satisfying a predicate function within a huge collection	Not Applicable	Aggregate … In … Into LongCount()
Max	Find out the maximum value within a collection	Not Applicable	Aggregate … In … Into Max()
Min	Find out the minimum value existing within a collection	Not Applicable	Aggregate … In … Into Min()
Sum	Find out the sum of a values within a collection	Not Applicable	Aggregate … In … Into Sum()

Quantifier Operations

These operators return a Boolean value i.e. True or False when some or all elements within a sequence satisfy a specific condition.

Show Examples

Operator	Description	C# Query Expression Syntax	VB Query Expression Syntax
All	Returns a value 'True' if all elements of a sequence satisfy a predicate condition	Not Applicable	Aggregate … In … Into All(…)
Any	Determines by searching a sequence that whether any element of the same satisfy a specified condition	Not Applicable	Aggregate … In … Into Any()
Contains	Returns a 'True' value if finds that a specific element is there in a sequence if the sequence doe not contains that specific element , 'false' value is returned	Not Applicable	Not Applicable

Partition Operators

Divide an input sequence into two separate sections without rearranging the elements of the sequence and then returning one of them

Operator	Description	C# Query Expression Syntax	VB Query Expression Syntax
Skip	Skips some specified number of elements within a sequence and returns the remaining ones	Not Applicable	Skip
SkipWhile	Same as that of Skip with the only exception that number of elements to skip are specified by a Boolean condition	Not Applicable	Skip While
Take	Take a specified number of elements from a sequence and skip the remaining ones	Not Applicable	Take
TakeWhile	Same as that of Take except the fact that number of elements to take are specified by a Boolean condition	Not Applicable	Take While

Generation Operations

A new sequence of values is created by generational operators

Operator	Description	C# Query Expression Syntax	VB Query Expression Syntax
DefaultIfEmpty	When applied to an empty sequence, generate a default element within a sequence	Not Applicable	Not Applicable
Empty	Returns an empty sequence of values and is the most simplest generational operator	Not Applicable	Not Applicable
Range	Generates a collection having a sequence of integers or numbers	Not Applicable	Not Applicable
Repeat	Generates a sequence containing repeated values of a specific length	Not Applicable	Not Applicable

Conversions

The operators change the type of input objects and are used in a diverse range of applications.

Operator	Description	C# Query Expression Syntax	VB Query Expression Syntax
AsEnumerable	Returns the input typed as IEnumerable<T>	Not Applicable	Not Applicable
AsQueryable	A (generic) IEnumerable is converted to a (generic)	Not Applicable	Not

	IQueryable		Applicable
Cast	Performs casting of elements of a collection to a specified type	Use an explicitly typed range variable. Eg:from string str in words	From … As …
OfType	Filters values on basis of their , depending on their capability to be cast to a particular type	Not Applicable	Not Applicable
ToArray	Forces query execution and does conversion of a collection to an array	Not Applicable	Not Applicable
ToDictionary	On basis of a key selector function set elements into a Dictionary<TKey, TValue> and forces execution of a LINQ query	Not Applicable	Not Applicable
ToList	Forces execution of a query by converting a collection to a List<T>	Not Applicable	Not Applicable
ToLookup	Forces execution of a query and put elements into a Lookup<TKey, TElement> on basis of a key selector function	Not Applicable	Not Applicable

Element Operators

Except the DefaultIfEmpty, all the rest eight standard query element operators return a single element from a collection

Operator	Description	C# Query Expression Syntax	VB Query Expression Syntax
ElementAt	Returns an element present within a specific index in a collection	Not Applicable	Not Applicable
ElementAtOrDefault	Same as ElementAt except of the fact that it also returns a default value in case the specific index is out of range	Not Applicable	Not Applicable
First	Retrieves the first element within a collection or the first element satisfying a specific condition	Not Applicable	Not Applicable
FirstOrDefault	Same as First except the fact that it also returns a default value in case there is no existence of such elements	Not Applicable	Not Applicable
Last	Retrieves the last element present in a collection or the last element satisfying a specific condition	Not Applicable	Not Applicable

LastOrDefault	Same as Last except the fact that it also returns a default value in case there is no existence of any such element	Not Applicable	Not Applicable
Single	Returns the lone element of a collection or the lone element that satisfy a certain condition	Not Applicable	Not Applicable
SingleOrDefault	Same as Single except that it also returns a default value if there is no existence of any such lone element	Not Applicable	Not Applicable
DefaultIfEmpty	Returns a default value if the collection or list is empty or null	Not Applicable	Not Applicable

LINQ Query Syntax

There are two basic ways to write a LINQ query to IEnumerable collection or IQueryable data sources.

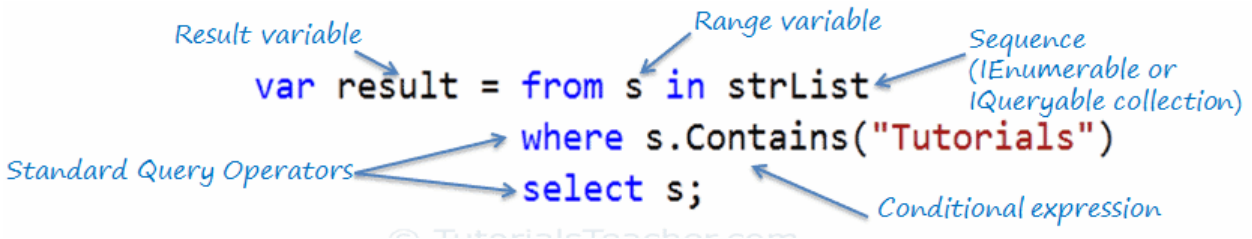
1. Query Syntax or Query Expression Syntax
2. Method Syntax or Method Extension Syntax or Fluent

Query Syntax

from <range variable> in <IEnumerable<T> or IQueryable<T> Collection>

<Standard Query Operators> <lambda expression>

<select or groupBy operator> <result formation>




```
strList=new List(Of String) from {“ram”,”vijay”,”hari”,”Tutorials}
```

Query syntax starts with a ***From*** clause followed by a ***Range*** variable. The From clause is structured like "**From** rangeV*ariableName* **in** *IEnumerablecollection*". In English, this means, from each object in the collection. It is similar to a foreach loop: foreach(Student s in studentList).

After the From clause, you can use different Standard Query Operators to filter, group, join elements of the collection. There are around 50 Standard Query Operators available in LINQ. In the above figure, we have used "where" operator (aka clause) followed by a condition. This condition is generally expressed using lambda expression.

LINQ query syntax always ends with a Select or Group clause. The Select clause is used to shape the data. You can select the whole object as it is or only some properties of it. In the above example, we selected the each resulted string elements.

Method Syntax

Method syntax (also known as fluent syntax) uses extension methods included in the **Enumerable** or **Queryable** static class, similar to how you would call the extension method of any class.

The compiler converts query syntax into method syntax at compile time.

Var Result= StrList.where(s=>s.contains(“ram”));

This is for c#

Var result=StrList.where(Funtion(s) s.contains(“ram”))

This is for vb.net

Method syntax comprises of extension methods and Lambda expression. Th extension method **Where()** is defined in the Enumerable class.

Simple Example:

Finding Numbers greater than 2 in given array With using If Statement

Step1:

First taken the Sequence Activity from Activity panel and changed the Display name as Finding_NumGreterThan_Two_WithUsing_If

Step 2:

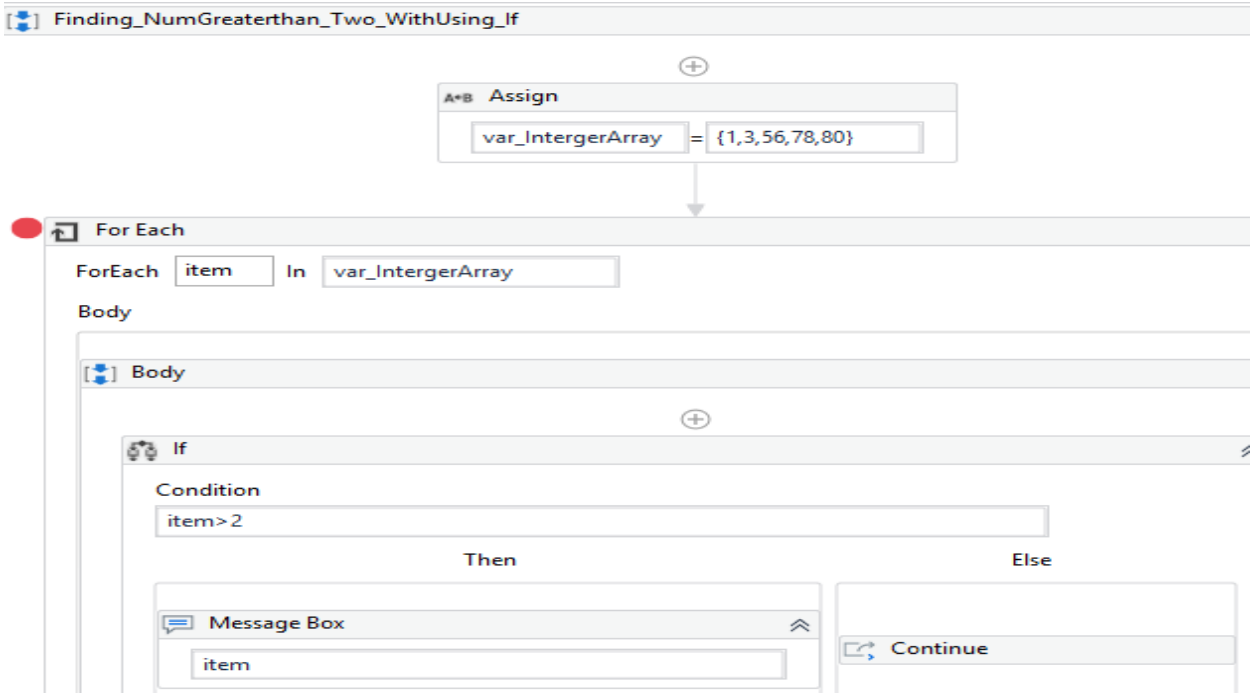
Take the Assign activity and create Array variable assigned values as wish and I given this values {1,3,36,78,80}

Step3:

Take for Each Activity for looping through each element in an array and the Type Argument is interger

Step4:

Take If activity and give condition greater then two in then body take the message box and give item and else body give continue the numbers less than two will we skipped and the loop continued



Finding Numbers greater than 2 in given array With using LINQ

Step1:

First taken the Sequence Activity from Activity panel and changed the Display name as Finding_NumGreterThan_Two_WithUsing_If

Step 2:

Take the Assign activity and create Array variable name var_IntegerArray assigned values as wish and I given this values {1,3,36,78,80}

Step3:

Take Assign Activity and write LINQ Query for a given array the numbers greater than two, the variable type should be Ienumerable type

var_IntegerArrray.Where(function(x) x>2)

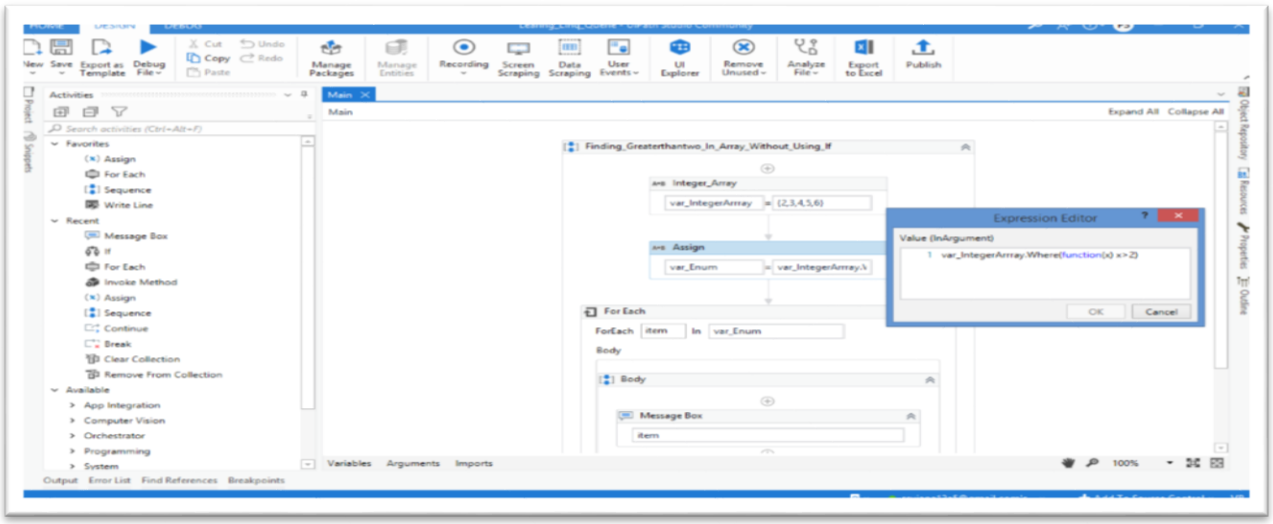
Function(x) – Input collection

X>2 -- output Boolean value

Function Enumerable.Where(Of Integer)(predicate As Func(Of Integer,Boolean)) As IEnumerable(Of Integer)

Step4:

Take for Each Activity for looping through each element in an array and the Type Argument is interger



Another Example

Step1: The Input Excel data

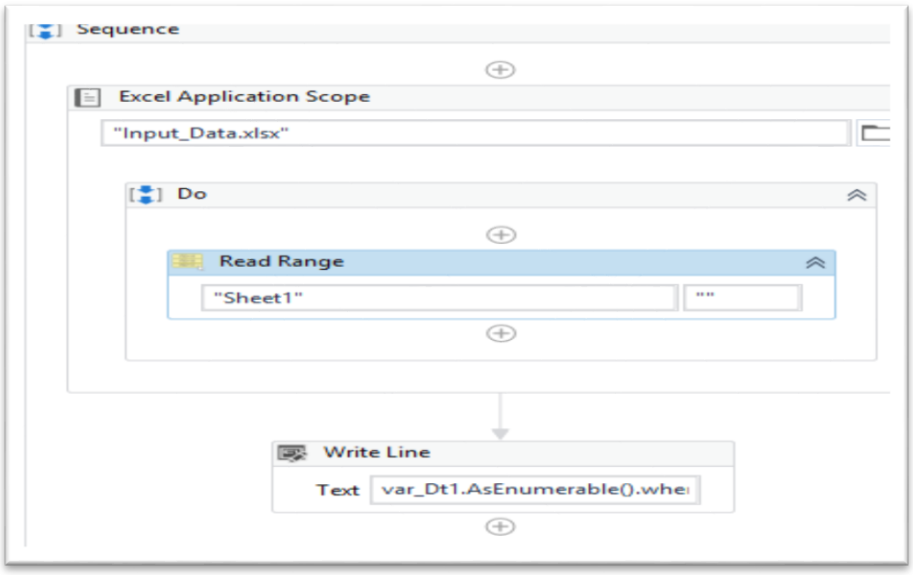
A	B	C	D
Seller	Order ID	Status	Amount
A	1	Shipped	700.00
A	2	Delivered	800.00
B	3	Shipped	300.00
C	4	Delivered	1000.00
B	5	Shipped	500.00
A	6	Delivered	400.00
C	7	Shipped	500.00
C	8	Shipped	400.00
B	9	Shipped	500.00
C	10	Delivered	460.00
A	11	Shipped	520.00
A	12	Delivered	500.00
B	13	Delivered	120.00
B	14	Delivered	1230.00
D	15	Shipped	1450.00
A	16	Shipped	500.00
B	17	Shipped	600.00
C	18	Shipped	70.00
D	19	Delivered	80.00
B	20	Shipped	800.00
A	21	Delivered	9000.00
B	22	Delivered	1000.00
C	23	Shipped	200.00
D	24	Shipped	1200.00

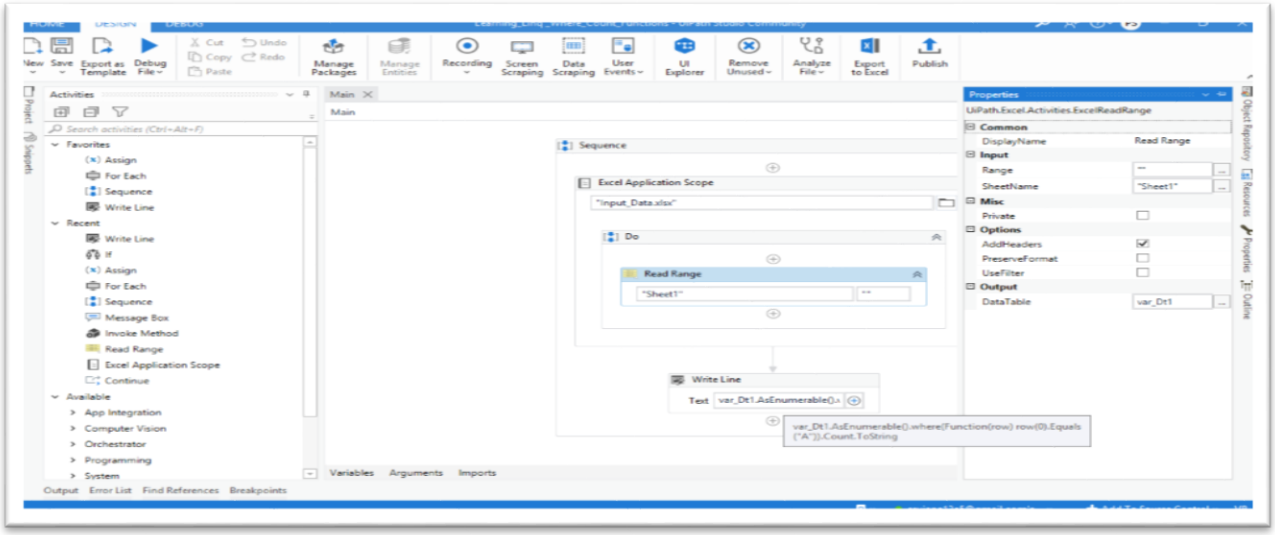
In order to read this excel we have to take Read Range Activity in the Excel application scope Activity

In read range Activity we have give sheet name and range and should create variable in output var_Dt1

Take write line activity and give LINQ query

var_Dt1.AsEnumerable().where(Function(row) row(0).Equals("A")).Count.ToString





Sum of all row in Amount Coloumn

var_Dt1.AsEnumerable().Sum(Function(row) Cint(row.item("Amount"))).ToString

Sum of Seller A

var_Dt1.AsEnumerable().Where(Function(row) Cstr(row.item("Seller")).Equals("A")).

Sum(Function(row) Cint(row.item(“Amount))).ToString

Maximum value of Seller A

var_Dt1.AsEnumerable().Where(Function(row) Cstr(row.item(“Seller”)).Equals(“A”)).

Max(Function(row) Cint(row.item(“Amount”)).ToString

Minimum value of Seller A

var_Dt1.AsEnumerable().Where(Function(row) Cstr(row.item(“Seller”)).Equals(“A”)).

Min(Function(row) Cint(row.item(“Amount”)).ToString

Find The Maximum Amount Where Seller is A And Status is Delivered

Var_Dt1.AsEnumerable().Where(Function(row) Cstr(row.item(“Seller”)).Equals(“A”)
And Cstr(row.item(“Status”)).Equals(“Delivered”)).Max(Function(row)
Cint(row.item(“Amount”)).ToString

Find The Maximum Amount Where Seller is A And Status is Delivered

Var_Dt1.AsEnumerable().Where(Function(row) Cstr(row.item(“Seller”)).Equals(“A”)
And Cstr(row.item(“Status”)).Equals(“Delivered”)).Max(Function(row)
Cint(row.item(“Amount”)).ToString

Find The Minimum Amount Where Seller is B And Status is Shipped

Var_Dt1.AsEnumerable().Where(Function(row) Cstr(row.item(“Seller”)).Equals(“B”)
And Cstr(row.item(“Status”)).Equals(“Shipped”)).Min(Function(row)
Cint(row.item(“Amount”)).ToString

Compare to Data tables

Get rows that exists in dtTable1 but not in dtTable2

dtDifference= dtTable1.AsEnumerable().Except(dtTable2.AsEnumerable, DataRowComparer.Default).CopyToDataTable

Get rows that exists in dtTable2 but not in dtTable1

dtDifference= dtTable2.AsEnumerable().Except(dtTable1.AsEnumerable, System.Data.DataRowComparer.Default).CopyToDataTable

System.Data ---Namespace

DataRowComparator----Class

Default----Property

In place of Except we can write Union and Intersect

Intersect means common in both the tables

The union of a table1with a table2 is the set of elements\row\coloumn that are in either table1 or table2