|  |
| --- |
| Photo displaying partial image of two pie charts on a canvas-textured page |
| IronXL.Excel Guide  Version 2019.5.2 |
| |  |  |  | | --- | --- | --- | | IronSoftware | 1/23/20 | Ahmed Aboelmagd | |

Contents

[Introduction 2](#_Toc35941479)

[Document Organization 2](#_Toc35941480)

[Chapter 1: Install IronXL.Excel 2](#_Toc35941481)

[Install using NuGet 2](#_Toc35941482)

[Using NuGet Package Manager 3](#_Toc35941483)

[Using NuGet Package Console manager 4](#_Toc35941484)

[Chapter 2: Basic Operations 4](#_Toc35941485)

[1- Sample: HelloWorld Console Application 4](#_Toc35941486)

[2. Create New Excel File you can create new excel file using IronXL as follow 7](#_Toc35941487)

[3. Open (CSV, XML, JSON List) as Workbook 7](#_Toc35941488)

[3.1. Open CSV file 7](#_Toc35941489)

[3.2. open XML File 8](#_Toc35941490)

[3.4. open JSON List as workbook 8](#_Toc35941491)

[4. Save and Export 11](#_Toc35941492)

[4.1. Save to “.xlsx” 11](#_Toc35941493)

[4.2. Save to csv “.csv” 11](#_Toc35941494)

[4.3. Save to JSON “.json” 12](#_Toc35941495)

[4.4. Save to XML “.xml” 12](#_Toc35941496)

[Chapter 3: Advanced Sheet Operations 12](#_Toc35941497)

[1- Sum Example 12](#_Toc35941498)

[2- AVG Example 13](#_Toc35941499)

[3- Count Example 13](#_Toc35941500)

[4- Max Example 13](#_Toc35941501)

[5- Min Example 14](#_Toc35941502)

[6- Order cells Example 14](#_Toc35941503)

[7- If Condition Example 14](#_Toc35941504)

[8- Trim Example 14](#_Toc35941505)

[Chapter 4: Working with workbooks contains more than one sheet 15](#_Toc35941506)

[1- Read Data from multiple sheets in the same workbook 15](#_Toc35941507)

[2- Add New Sheet to a workbook 15](#_Toc35941508)

[Chapter 5: Integration with EF/Database 16](#_Toc35941509)

[1- Fill Excel sheet with data from Database 16](#_Toc35941510)

[2- Fill Database with data from Excel sheet 16](#_Toc35941511)

[Summary 17](#_Toc35941512)

[Author 17](#_Toc35941513)

# Introduction

This document should describe main functions to use the IronXL library and it will introduce basic sample on various library functions. you can visit <https://ironsoftware.com/csharp/excel/> for more information. You can download sample project from GitHub (https://github.com/magedo93/IronSoftware.git)

# Document Organization

* Chapter 1 Install IronXL.Excel: this part describes How to install IronXL.Excel to an existing project.
* Chapter 2 Basic Operations: this part describes basic operation with excel create or Open workbook, select sheet, select cell save the workbook
* Chapter 3 Advanced Sheet Operations: this part describes how to different manipulation capabilities like adding headers or footers, mathematical operations files, and other features.
* Summery   
  brief conclusion about what we have learned in this document
* About author   
  brief about document author

# Chapter 1: Install IronXL.Excel

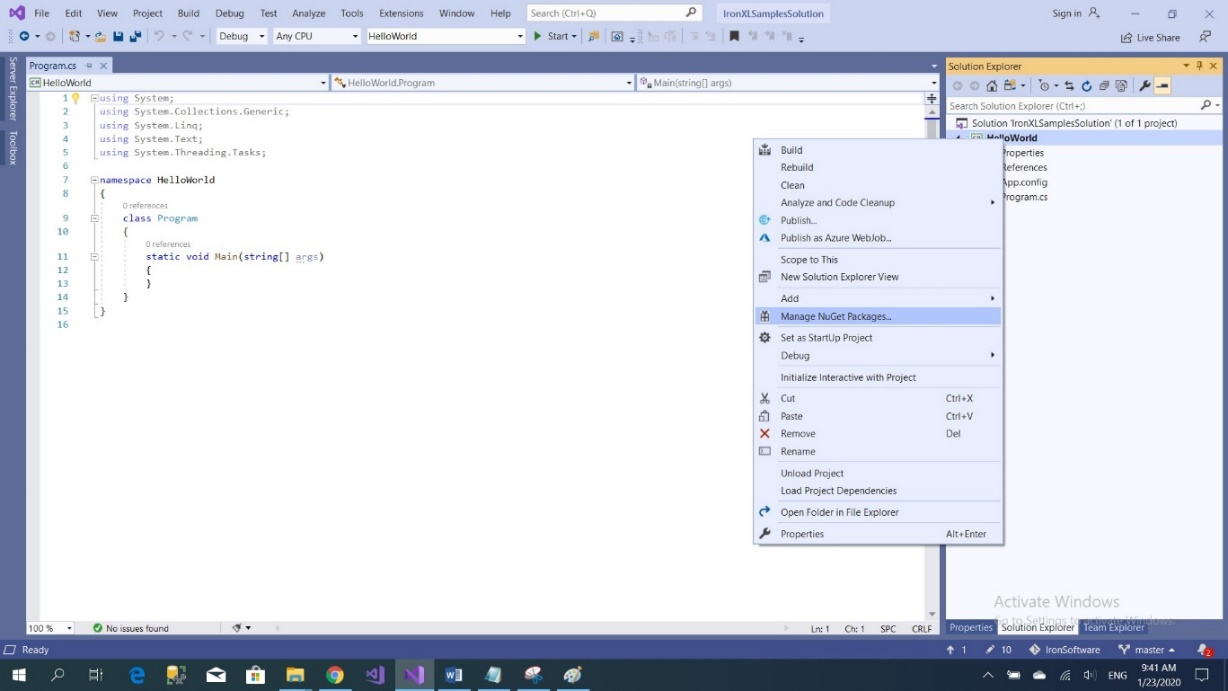
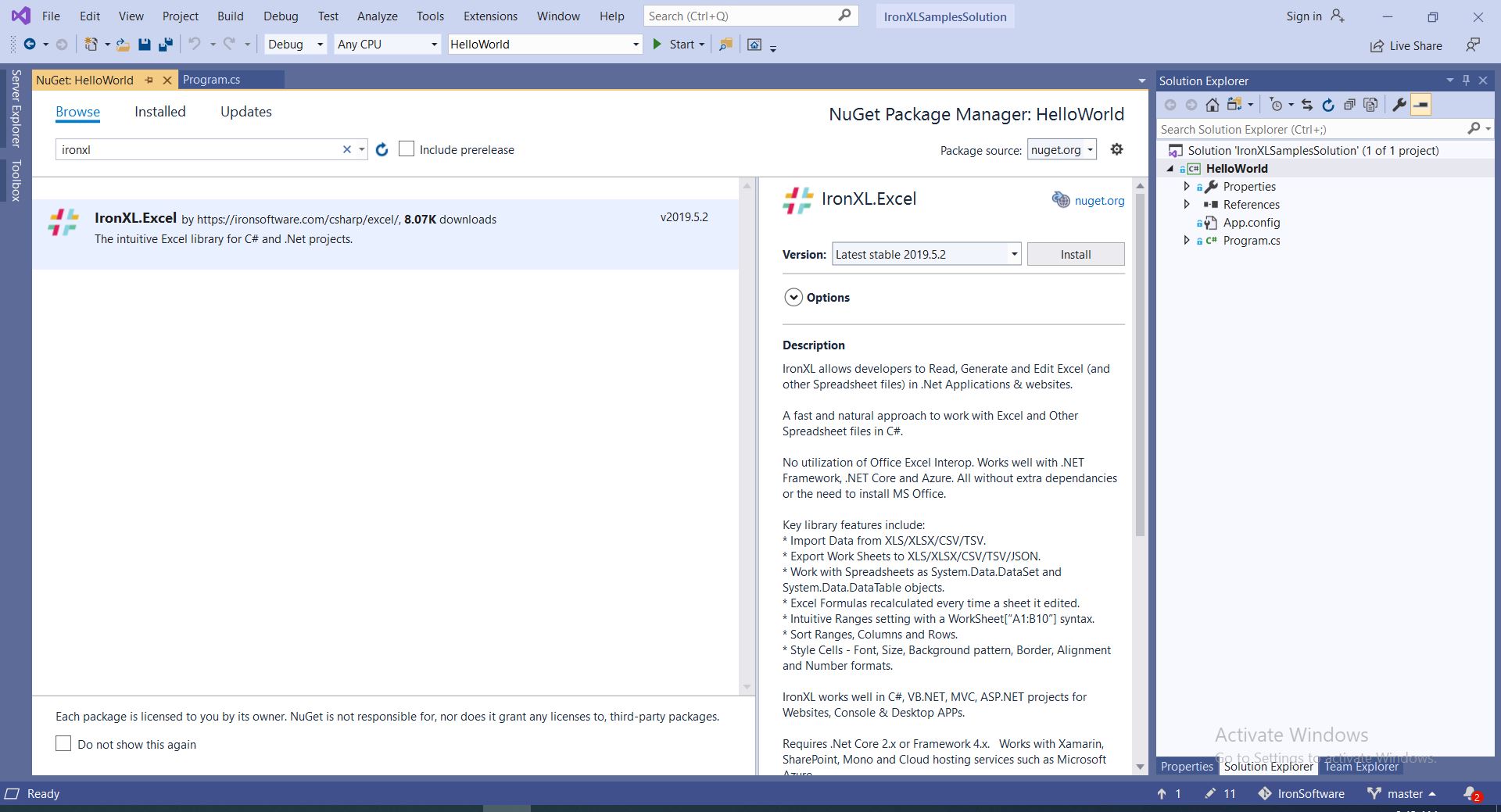
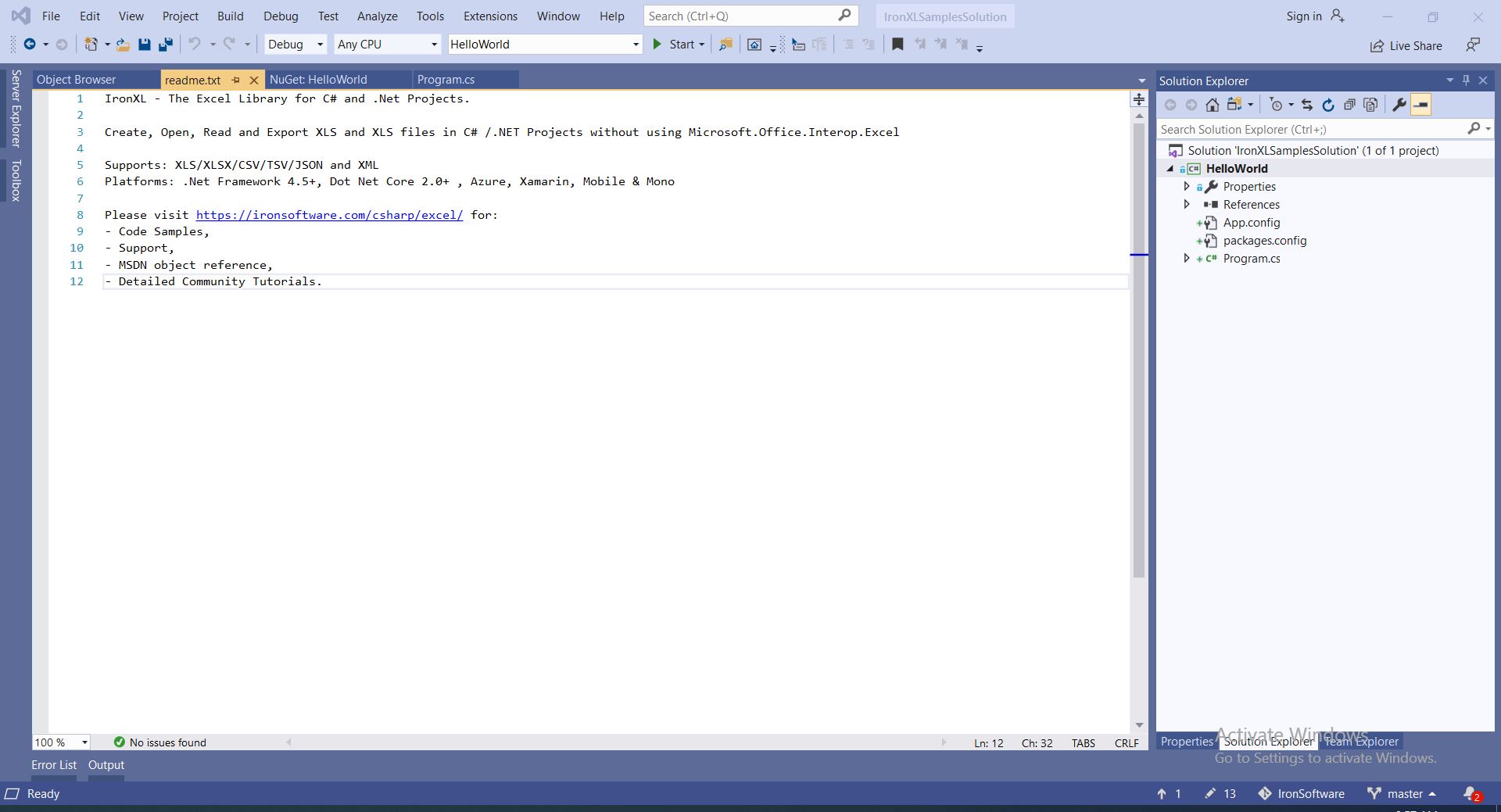
## IronXL.Excel can be installed and used on all of the .NET project types like windows application, ASP.NET MVC and .Net Core Application.

## To add IronXL.Excel library to the project we have two ways, from Visual studio editor install using NuGet or command line using package console manager as following: -

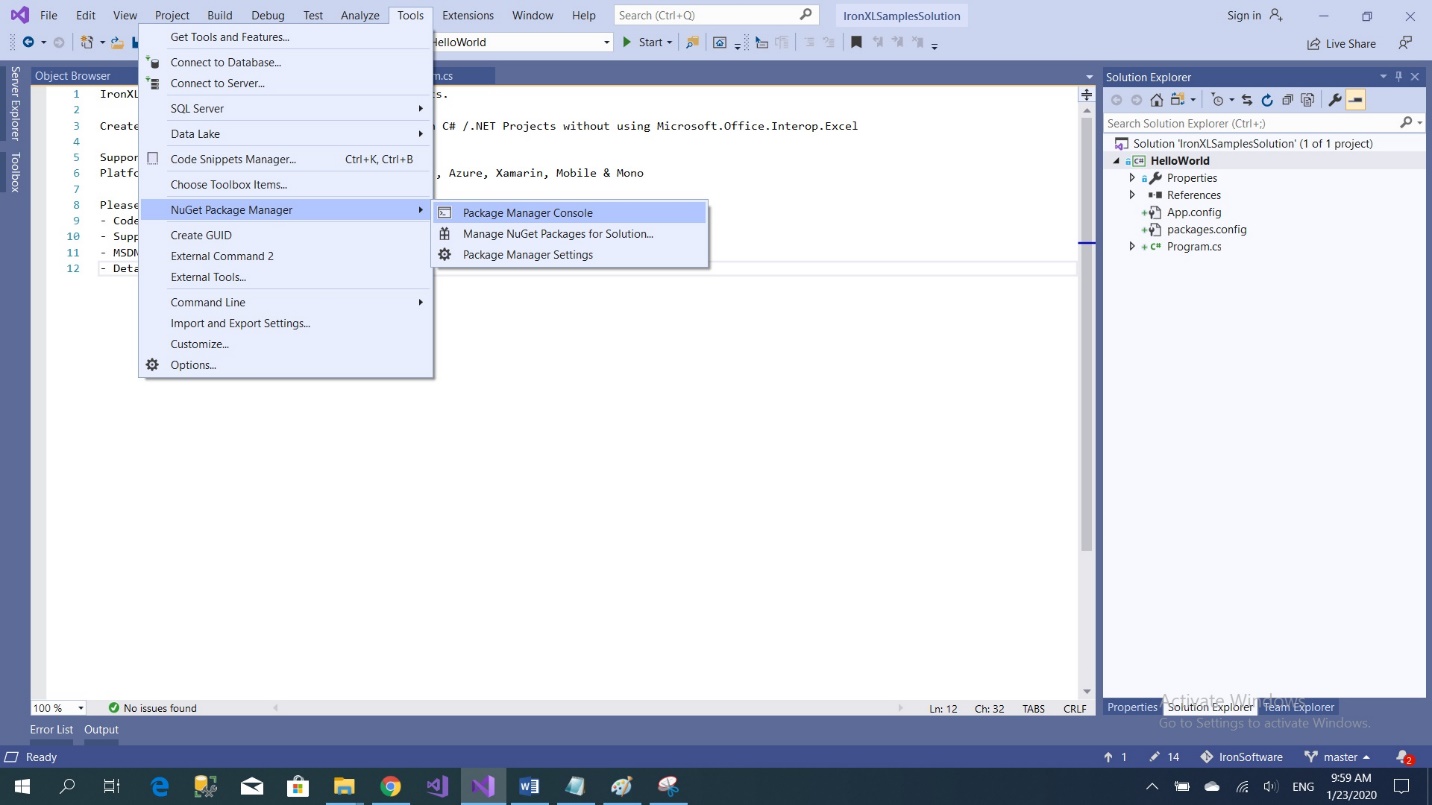
## Install using NuGet

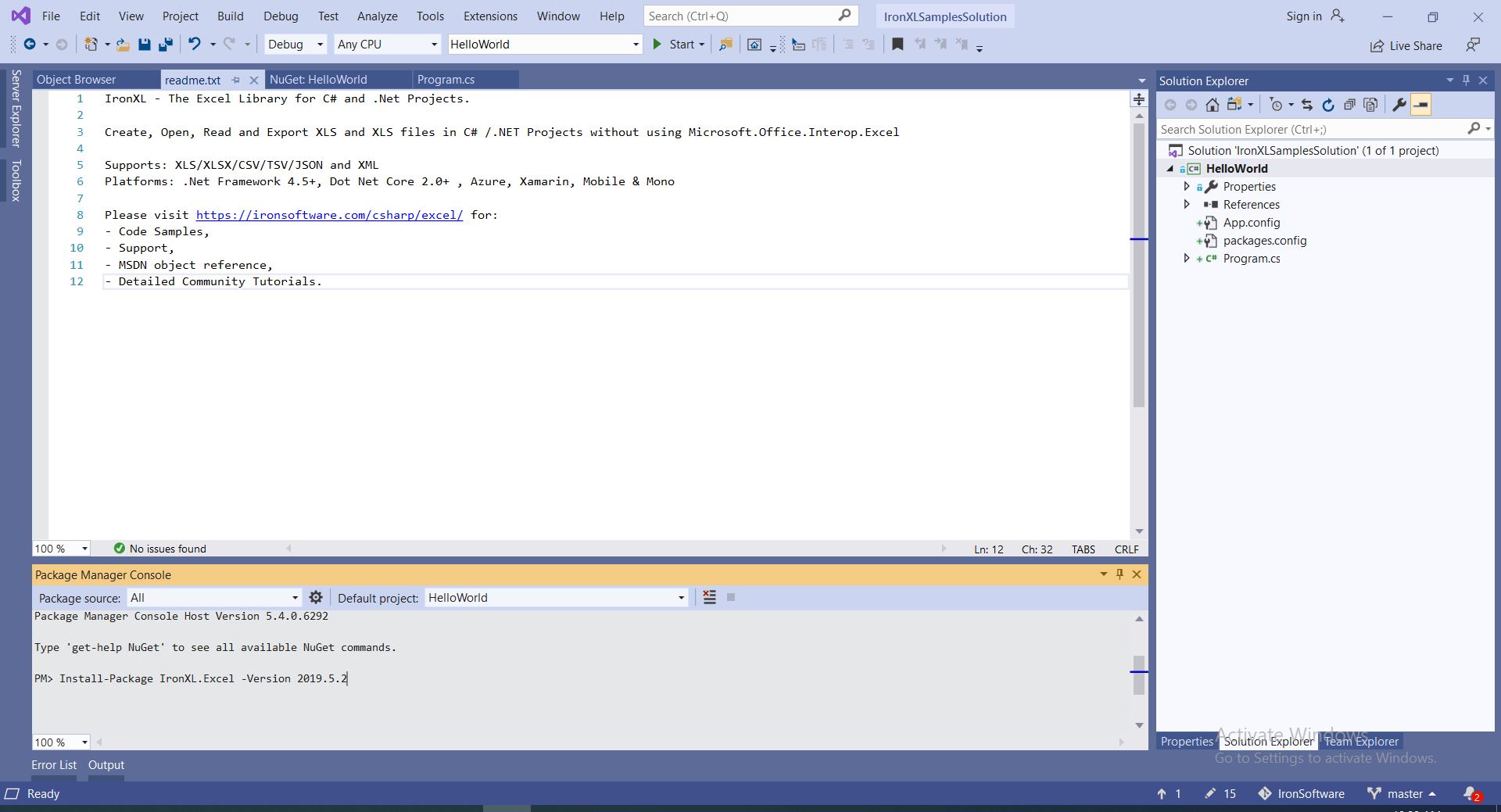
To add IronXL.Excel library to our project using NuGet we can do it using a visualized interface (NuGet Package Manager) or by command using Package Manager Console as follows: -

Using NuGet Package Manager

1. Using mouse -> right click on project name -> Select manage NuGet Package  
   
2. From brows tab -> search for IronXL.Excel -> Install   
   
3. And we are Done  
   

### Using NuGet Package Console manager

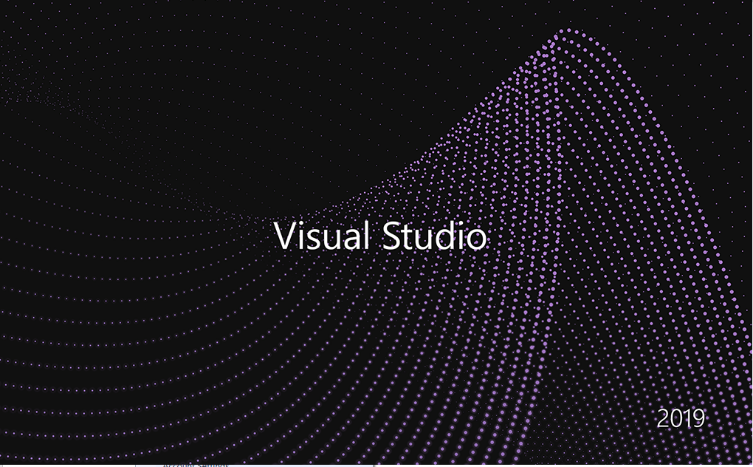
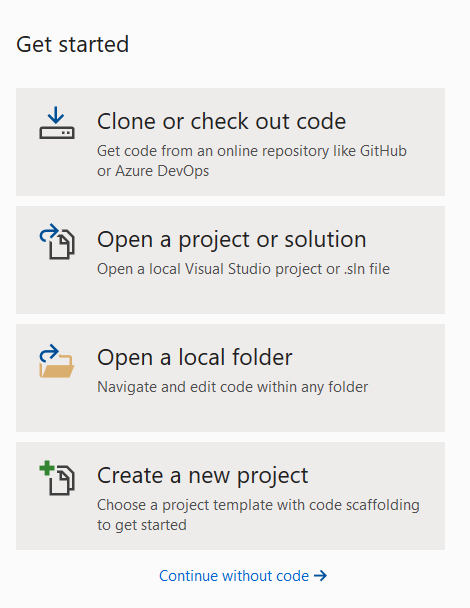
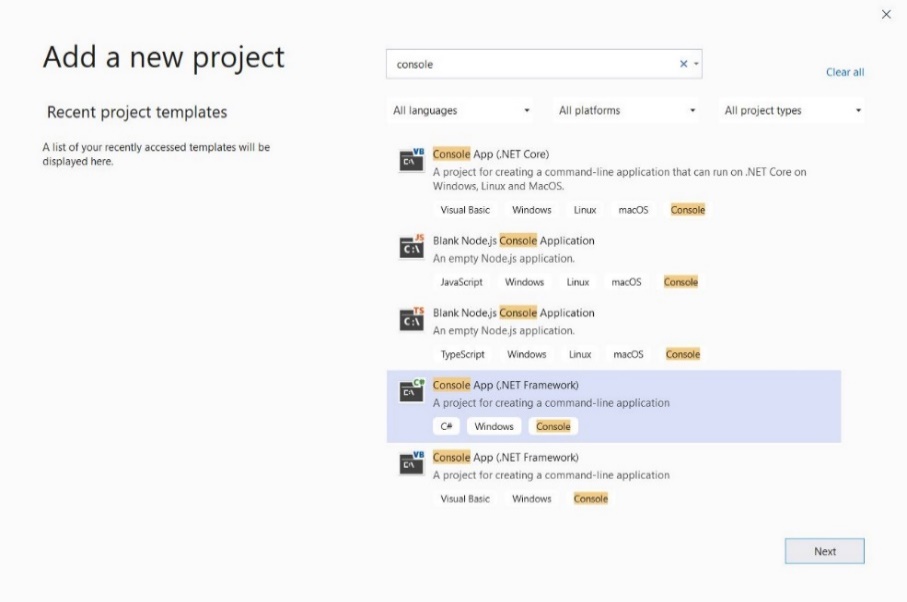
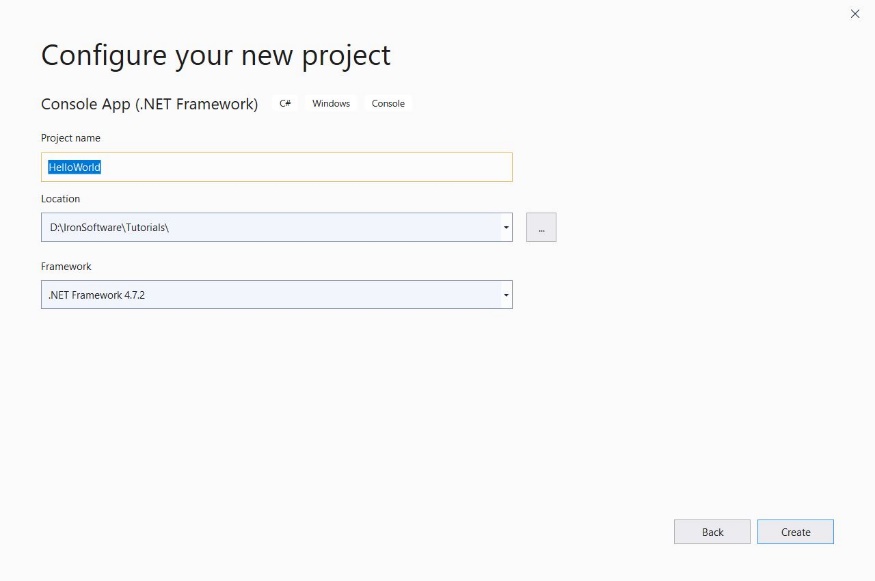
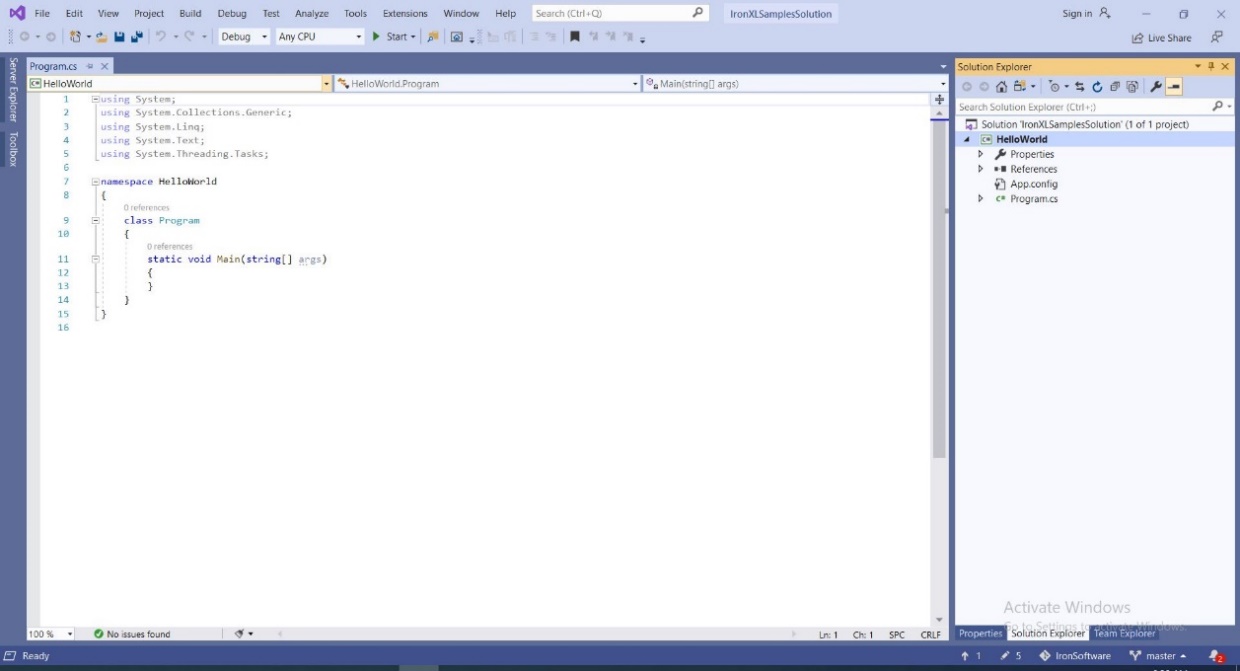
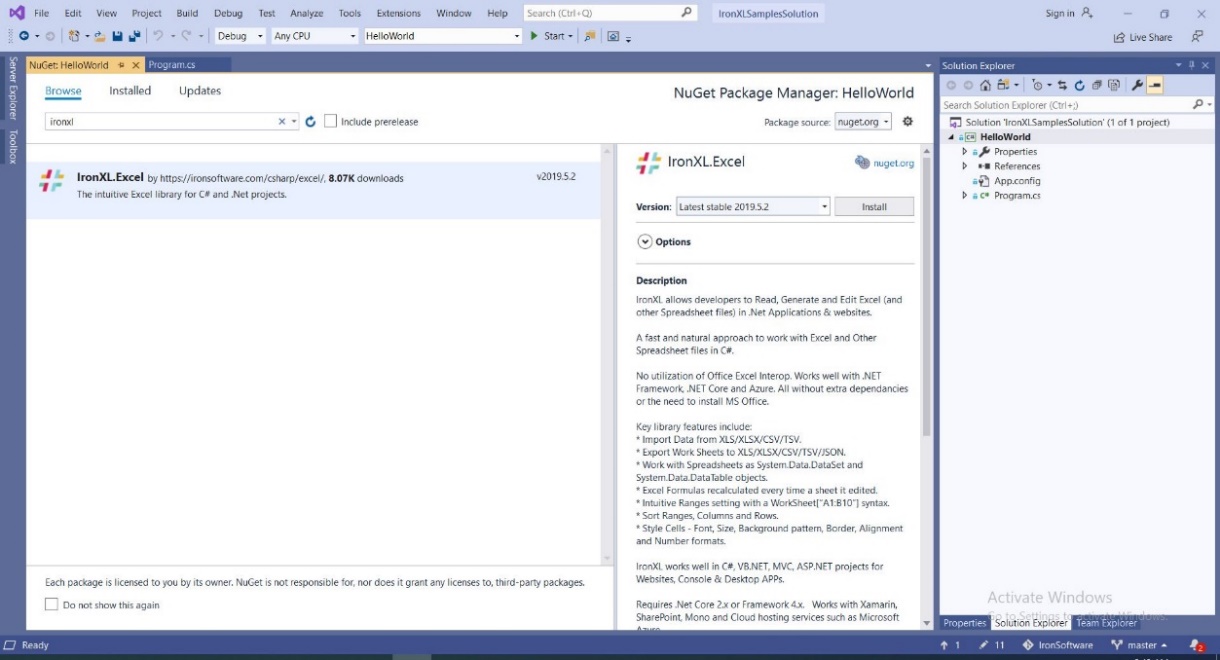
1. From tools -> NuGet Package Manager -> Package Manager Console  
   
2. Run command -> Install-Package IronXL.Excel -Version 2019.5.2



# Chapter 2: Basic Operations

## Sample: HelloWorld Console Application

Follow coming steps to create HelloWorld Project

* 1. Open visual studio   
     
  2. Choose Create new project   
     
  3. Choose Console App (.NET framework)  
     
  4. Give our sample name “HelloWorld” and click create   
     
  5. Now we have console application created   
     
  6. Add IronXL.Excel => click install 
  7. Add our first few lines that reads 1st cell in 1st sheet in excel file and print it

|  |
| --- |
| static void Main(string[] args)  {  var workbook = IronXL.WorkBook.Load($@"{Directory.GetCurrentDirectory()}\Files\HelloWorld.xlsx");  var sheet = workbook.WorkSheets.First();  var cell = sheet["A1"].StringValue;  Console.WriteLine(cell);  } |

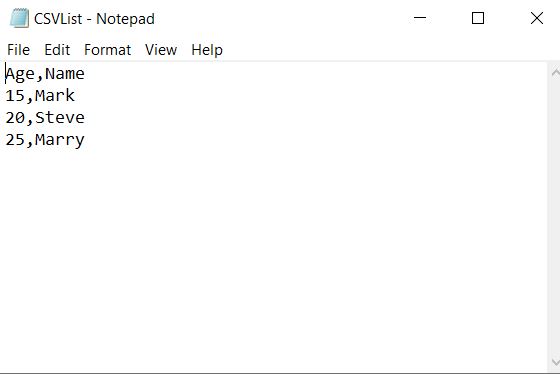
## Create New Excel File you can create new excel file using IronXL as follow

|  |
| --- |
| static void Main(string[] args)  {  var newXLFile = WorkBook.Create(ExcelFileFormat.XLSX);  newXLFile.Metadata.Title = "IronXL New File";  var newWorkSheet = newXLFile.CreateWorkSheet("1stWorkSheet");  newWorkSheet["A1"].Value = "Hello World";  newWorkSheet["A2"].Style.BottomBorder.SetColor("#ff6600");  newWorkSheet["A2"].Style.BottomBorder.Type = IronXL.Styles.BorderType.Dashed;  } |

## Open (CSV, XML, JSON List) as Workbook

### Open CSV file

* + 1. Create new text file and add to its list of names and ages as follow then save it as CSVList.csv



* + 1. Code snippet should be like this

|  |
| --- |
| static void Main(string[] args)  {  var workbook = IronXL.WorkBook.Load($@"{Directory.GetCurrentDirectory()}\Files\CSVList.csv");  var sheet = workbook.WorkSheets.First();  var cell = sheet["A1"].StringValue;  Console.WriteLine(cell);  } |

### open XML File

create an XML file that contains countries list as following   
root element “countries” that have children elements “country” and each country has properties that define the country like code, continent, etc.

|  |
| --- |
| <?xml version="1.0" encoding="utf-8"?>  <countries xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:xsd="http://www.w3.org/2001/XMLSchema">  <country code="ae" handle="united-arab-emirates" continent="asia" iso="784">United Arab Emirates</country>  <country code="gb" handle="united-kingdom" continent="europe" alt="England Scotland Wales GB UK Great Britain Britain Northern" boost="3" iso="826">United Kingdom</country>  <country code="us" handle="united-states" continent="north america" alt="US America USA" boost="2" iso="840">United States</country>  <country code="um" handle="united-states-minor-outlying-islands" continent="north america" iso="581">United States Minor Outlying Islands</country>  </countries> |

* 1. and now copy the following code snippet that open XML as workbook

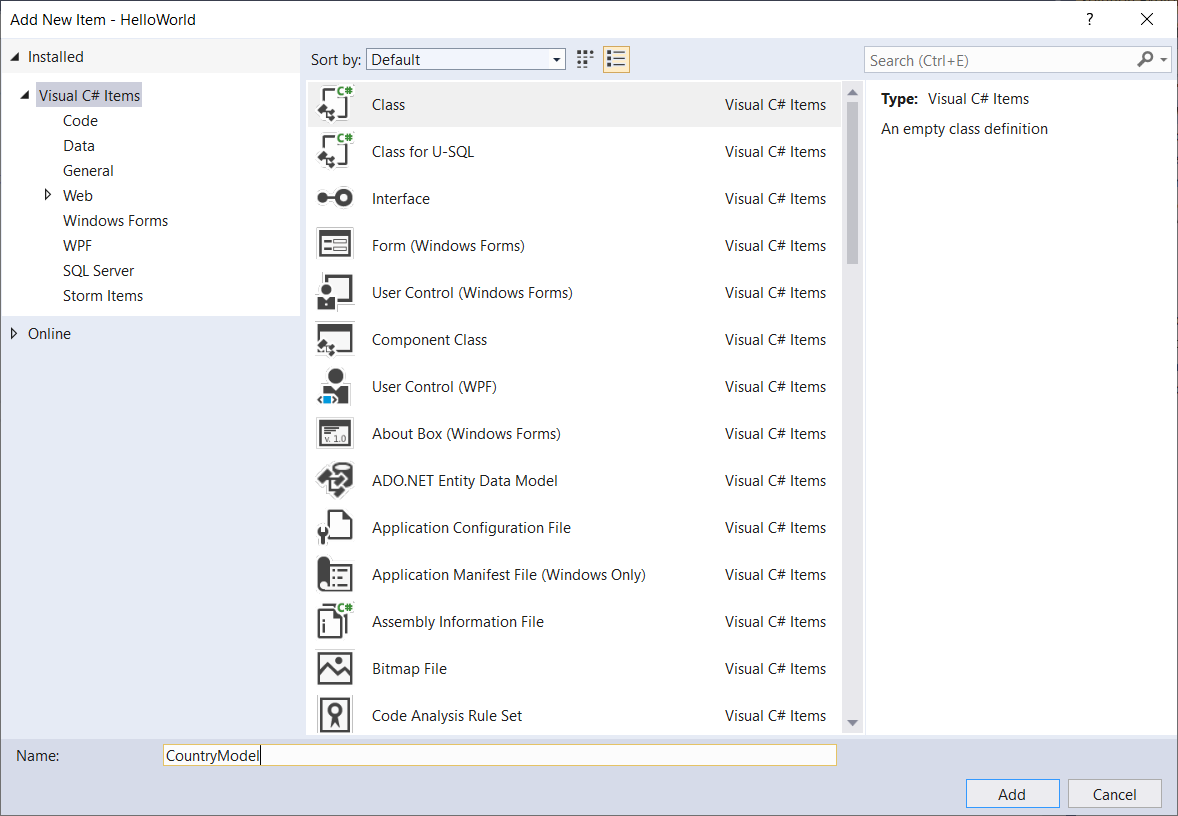
|  |
| --- |
| static void Main(string[] args)  {  var xmldataset = new DataSet();  xmldataset.ReadXml($@"{Directory.GetCurrentDirectory()}\Files\CountryList.xml");  var workbook = IronXL.WorkBook.Load(xmldataset);  var sheet = workbook.WorkSheets.First();  } |

### open JSON List as workbook

create JSON country list as following

|  |
| --- |
| [  {  "name": "United Arab Emirates",  "code": "AE"  },  {  "name": "United Kingdom",  "code": "GB"  },  {  "name": "United States",  "code": "US"  },  {  "name": "United States Minor Outlying Islands",  "code": "UM"  }  ] |

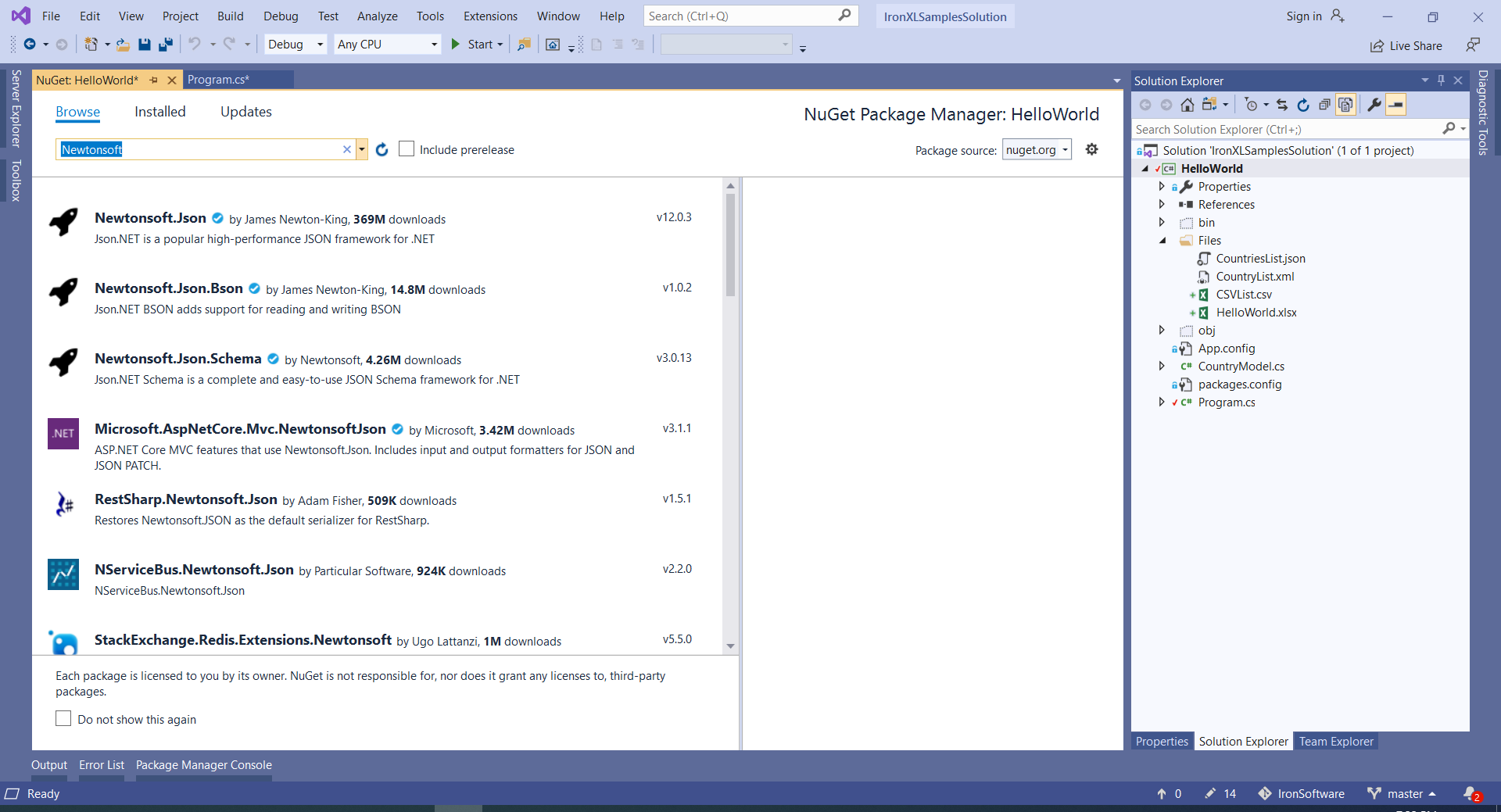
* + 1. create country model that will map to JSON



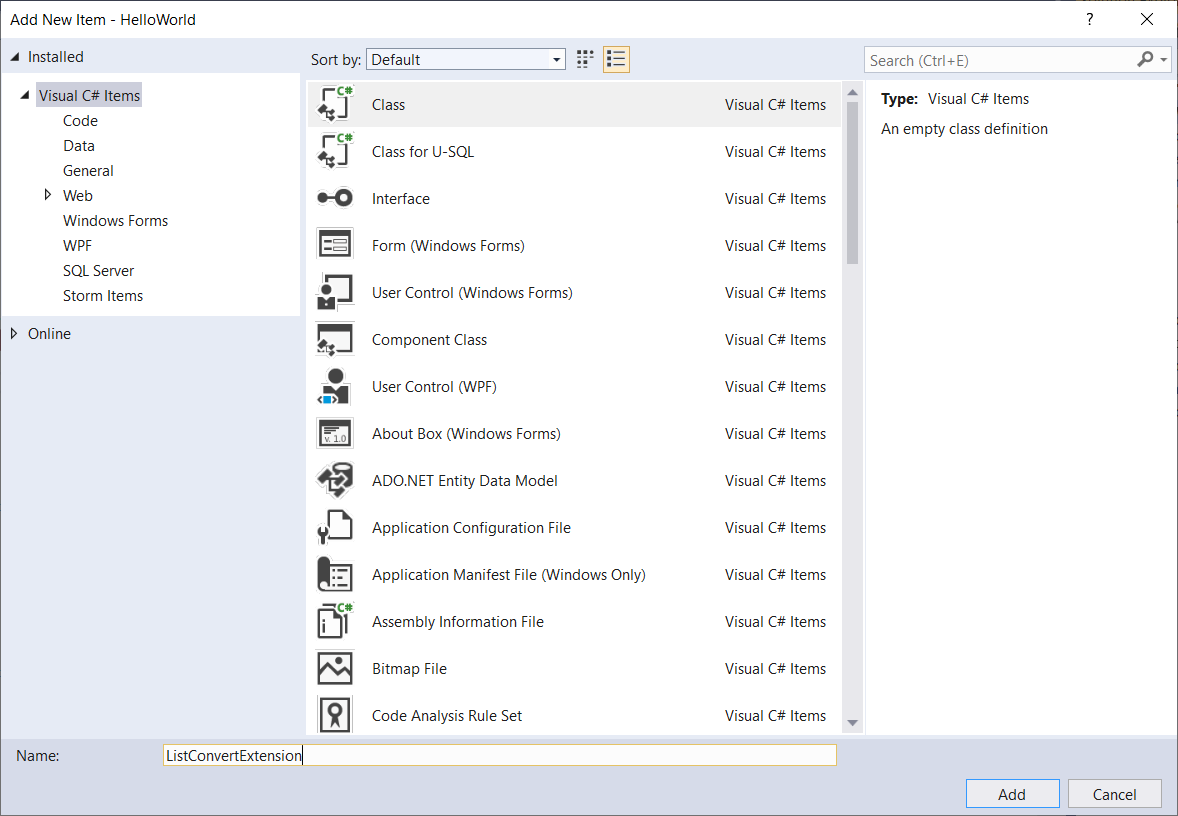
* + 1. and here is the class code snippet

|  |
| --- |
| public class CountryModel  {  public string name { get; set; }  public string code { get; set; }  } |

* + 1. Add Newtonsoft library to convert JSON to list of country models



* + 1. To convert list to dataset we have to create new extension for list as following, first add extension class with the name “ListConvertExtension”



* + 1. Then add this code snippet to it

|  |
| --- |
| public static class ListConvertExtension  {  public static DataSet ToDataSet<T>(this IList<T> list)  {  Type elementType = typeof(T);  DataSet ds = new DataSet();  DataTable t = new DataTable();  ds.Tables.Add(t);  //add a column to table for each public property on T  foreach (var propInfo in elementType.GetProperties())  {  Type ColType = Nullable.GetUnderlyingType(propInfo.PropertyType) ?? propInfo.PropertyType;  t.Columns.Add(propInfo.Name, ColType);  }  //go through each property on T and add each value to the table  foreach (T item in list)  {  DataRow row = t.NewRow();  foreach (var propInfo in elementType.GetProperties())  {  row[propInfo.Name] = propInfo.GetValue(item, null) ?? DBNull.Value;  }  t.Rows.Add(row);  }  return ds;  }  } |

* + 1. And finally load this dataset as workbook

|  |
| --- |
| static void Main(string[] args)  {  var jsonFile = new StreamReader($@"{Directory.GetCurrentDirectory()}\Files\CountriesList.json");  var countryList = Newtonsoft.Json.JsonConvert.DeserializeObject<CountryModel[]>(jsonFile.ReadToEnd());  var xmldataset = countryList.ToDataSet();  var workbook = IronXL.WorkBook.Load(xmldataset);  var sheet = workbook.WorkSheets.First();  } |

## Save and Export

We can save or export excel file to multiple files formats like (“.xlsx”,”.csv”,”.html”) using one of the following commands.

### Save to “.xlsx”

To Save to “.xlsx” use saveAs function

|  |
| --- |
| static void Main(string[] args)  {  var newXLFile = WorkBook.Create(ExcelFileFormat.XLSX);  newXLFile.Metadata.Title = "IronXL New File";  var newWorkSheet = newXLFile.CreateWorkSheet("1stWorkSheet");  newWorkSheet["A1"].Value = "Hello World";  newWorkSheet["A2"].Style.BottomBorder.SetColor("#ff6600");  newWorkSheet["A2"].Style.BottomBorder.Type = IronXL.Styles.BorderType.Dashed;  newXLFile.SaveAs($@"{Directory.GetCurrentDirectory()}\Files\HelloWorld.xlsx");  } |

### Save to csv “.csv”

To save to “.csv” we can use SaveAsCsv and pass to it 2 parameters 1st parameter the file name and path the 2nd parameter is the delimiter like (“,” or “|” or “:”)

|  |
| --- |
| newXLFile.SaveAsCsv([$@"{Directory.GetCurrentDirectory()}\Files\HelloWorld.csv",delimiter:"|")](mailto:$@%22%7bDirectory.GetCurrentDirectory()%7d\Files\HelloWorld.csv%22,delimiter:%22|%22)); |

### Save to JSON “.json”

To save to Json “.json” use SaveAsJson as follow

|  |
| --- |
| newXLFile.SaveAsJson($@"{Directory.GetCurrentDirectory()}\Files\HelloWorldJSON.json"); |

The result file should be as follow

|  |
| --- |
| [  [  "Hello World"  ],  [  ""  ]  ] |

### Save to XML “.xml”

To save to xml use SaveAsXml as follow

|  |
| --- |
| newXLFile.SaveAsXml([$@"{Directory.GetCurrentDirectory()}\Files\HelloWorldXML.XML")](mailto:$@%22%7bDirectory.GetCurrentDirectory()%7d\Files\HelloWorldXML.XML%22)); |

Result should be like the following

|  |
| --- |
| <?xml version="1.0" standalone="yes"?>  <\_x0031\_stWorkSheet>  <\_x0031\_stWorkSheet>  <Column1 xsi:type="xs:string" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">Hello World</Column1>  </\_x0031\_stWorkSheet>  <\_x0031\_stWorkSheet>  <Column1 xsi:type="xs:string" xmlns:xs="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" />  </\_x0031\_stWorkSheet>  </\_x0031\_stWorkSheet> |

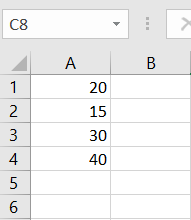
# Chapter 3: Advanced Sheet Operations

In this chapter we will go throw common excel functions like (SUM, AVG, Count)

I’ll write short code snippets for each function Let’s go.

## Sum Example

Let’s find the sum for this list, I created excel file and named it “Sum.xlsx” and add this list of numbers manually



|  |
| --- |
| var workbook = IronXL.WorkBook.Load($@"{Directory.GetCurrentDirectory()}\Files\Sum.xlsx");  var sheet = workbook.WorkSheets.First();  decimal sum = sheet["A2:A4"].Sum();  Console.WriteLine(sum); |

## AVG Example

Using same file, we can get the average as follow: -

|  |
| --- |
| var workbook = IronXL.WorkBook.Load($@"{Directory.GetCurrentDirectory()}\Files\Sum.xlsx");  var sheet = workbook.WorkSheets.First();  decimal avg = sheet["A2:A4"].Avg();  Console.WriteLine(avg); |

## Count Example

Using same file, we can get the number of elements in a sequence as follow: -

|  |
| --- |
| var workbook = IronXL.WorkBook.Load($@"{Directory.GetCurrentDirectory()}\Files\Sum.xlsx");  var sheet = workbook.WorkSheets.First();  decimal count = sheet["A2:A4"].Count();  Console.WriteLine(count); |

## Max Example

Using same file, we can get max value of range of cells as follow:

|  |
| --- |
| var workbook = IronXL.WorkBook.Load($@"{Directory.GetCurrentDirectory()}\Files\Sum.xlsx");  var sheet = workbook.WorkSheets.First();  decimal max = sheet["A2:A4"].Max ();  Console.WriteLine(max); |

– we can apply transform function to the result of max function as follow:

|  |
| --- |
| var workbook = IronXL.WorkBook.Load($@"{Directory.GetCurrentDirectory()}\Files\Sum.xlsx");  var sheet = workbook.WorkSheets.First();  bool max2 =sheet["A1:A4"].Max(c => c. IsFormula);  Console.WriteLine(count); |

This example write “false” in the console.

## Min Example

Using same file, we can get min value of range of cells as follow

|  |
| --- |
| var workbook = IronXL.WorkBook.Load($@"{Directory.GetCurrentDirectory()}\Files\Sum.xlsx");  var sheet = workbook.WorkSheets.First();  bool max2 =sheet["A1:A4"].Min();  Console.WriteLine(count); |

## Order cells Example

Using same file, we can order cells ascending or descending as follow

|  |
| --- |
| var workbook = IronXL.WorkBook.Load($@"{Directory.GetCurrentDirectory()}\Files\Sum.xlsx");  var sheet = workbook.WorkSheets.First();  sheet["A1:A4"].SortAscending(); //or use > sheet["A1:A4"].SortDescending(); to order descending  workbook.SaveAs("SortedSheet.xlsx"); |

## If Condition Example

Using same file, we can use Formula property to set or get cell’s formula.

4.1 – Set Cell‘s Formula as follow:

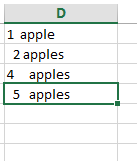
|  |
| --- |
| var workbook = IronXL.WorkBook.Load($@"{Directory.GetCurrentDirectory()}\Files\Sum.xlsx");  var sheet = workbook.WorkSheets.First();  int i = 1;  foreach(var cell in sheet["B1:B4"])  {  cell.Formula = "=IF(A" +i+ ">=20,\" Pass\" ,\" Fail\" )";  i++;  }  workbook.SaveAs($@"{Directory.GetCurrentDirectory()}\Files\NewExcelFile.xlsx"); |

4.2 – using the generated file form the previous example, we can get Cell’s Formula as follow:

|  |
| --- |
| var workbook = IronXL.WorkBook.Load($@"{Directory.GetCurrentDirectory()}\Files\NewExcelFile.xlsx");  var sheet = workbook.WorkSheets.First();  foreach(var cell in sheet["B1:B4"])  {  Console.WriteLine(cell.Formula);  }  Console.ReadKey(); |

## Trim Example

To apply trim function (to eliminate all extra spaces in cells) I added this column to the sum.xlsx file



And use this code

|  |
| --- |
| var workbook = IronXL.WorkBook.Load($@"{Directory.GetCurrentDirectory()}\Files\NewExcelFile.xlsx");  var sheet = workbook.WorkSheets.First();  int i = 1;  foreach (var cell in sheet["f1:f4"])  {  cell.Formula = "=trim(D"+i+")";  i++;  }  workbook.SaveAs("editedFile.xlsx"); |

Thus, you can apply formulas in the same way

# Chapter 4: Working with workbooks contains more than one sheet

In this chapter we will go through how to work with workbook that have more than one sheet

## Read Data from multiple sheets in the same workbook

I created xlsx file contains two sheets “Sheet1”,” Sheet2”

Till now we use WorkSheets.First() to work with the first sheet, this example to specify sheet name and work with it

|  |
| --- |
| var workbook = IronXL.WorkBook.Load($@"{Directory.GetCurrentDirectory()}\Files\testFile.xlsx");  WorkSheet sheet = workbook.GetWorkSheet("Sheet2");  var range = sheet["A2:D2"];  foreach(var cell in range)  {  Console.WriteLine(cell.Text);  } |

## Add New Sheet to a workbook

We can also add new sheet to a workbook as below

|  |
| --- |
| var workbook = IronXL.WorkBook.Load($@"{Directory.GetCurrentDirectory()}\Files\testFile.xlsx");  var newSheet = workbook.CreateWorkSheet("new\_sheet");  newSheet["A1"].Value = "Hello World";  workbook.SaveAs(@"F:\MY WORK\IronPackage\Xl tutorial\newFile.xlsx"); |

# Chapter 5: Integration with EF/Database

Let’s see how to export/ import data to/from Database

I created TestDb database that contains Country table with two columns Id (int, identity), CountryName(string)

## Fill Excel sheet with data from Database

Here we will create new sheet and fill it with data from Country Table

|  |
| --- |
| TestDbEntities dbContext = new TestDbEntities();  var workbook = IronXL.WorkBook.Load($@"{Directory.GetCurrentDirectory()}\Files\testFile.xlsx");  WorkSheet sheet = workbook.CreateWorkSheet("FromDb");  List<Country> countryList = dbContext.Countries.ToList();  sheet.SetCellValue(0, 0, "Id");  sheet.SetCellValue(0, 1, "Country Name");  int row = 1;  foreach (var item in countryList)  {  sheet.SetCellValue(row, 0, item.id);  sheet.SetCellValue(row, 1, item.CountryName);  row++;  }  workbook.SaveAs("FilledFile.xlsx"); |

## Fill Database with data from Excel sheet

In this example we will insert data to Country table in TestDb Database

|  |
| --- |
| TestDbEntities dbContext = new TestDbEntities ();  var workbook = IronXL.WorkBook.Load($@"{Directory.GetCurrentDirectory()}\Files\testFile.xlsx");  WorkSheet sheet = workbook.GetWorkSheet("Sheet3");  System.Data.DataTable dataTable = sheet.ToDataTable(true);  foreach (DataRow row in dataTable.Rows)  {  Country c = new Country();  c.CountryName = row[1].ToString();  dbContext.Countries.Add(c);  }  dbContext.SaveChanges(); |

# Summary

In this tutorial, we have learned how to (create, open, save) excel file using different ways and also we learned the basic operations that we can do on the file like getting (Sum, Avg., Count)

# Author



**Ahmed Aboelmagd** is a Full-stack experienced software engineer with 12+ years’ experience in IT and Software development, delivered 15+ successful project in many size scales from small to an enterprise for industries like (tourism, educational, manufacturing, etc.)

Code Samples prepared by Alaa Mohamed Nagih