

# Customization by Adding Your Own Code

## 1 CONTENTS

---

- 1 Introduction ..... 2
  - 1.1 Read these tutorials in order ..... 2
- 2 Adding New Files ..... 2
  - 2.1 Where Can You Add Files? ..... 2
  - 2.2 What Files Can You Add? ..... 3
  - 2.3 Why Add New Files? ..... 3
- 3 Adding Code to a Generated File ..... 3
- 4 End-to-End Example on Adding Your Own File and Code ..... 4
  - 4.1 Generate Code Using AspCoreGen 3.0 MVC Professional Plus ..... 4
  - 4.2 The Tutorial..... 4

# Customization by Adding Your Own Code

## 1 INTRODUCTION

---

This topic will show you how to add your own code to the ASP.NET Core MVC's generated code.

### 1.1 READ THESE TUTORIALS IN ORDER

1. Database Settings Tab
2. Code Settings Tab
3. UI Settings Tab
4. App Settings Tab
5. Selected Tables Tab
6. Selected Views Tab
7. Generating Code
8. The Generated Code for Database Tables/Views

Then follow these step-by-step instructions.

## 2 ADDING NEW FILES

---

Unlike the older versions, you can now add new files to any of the generated projects.

### 2.1 WHERE CAN YOU ADD FILES?

You can add files to the following generated projects:

1. Web Application Project (ASP.NET Core MVC)
2. Business Layer and Data Layer Project (Class Library).
3. Web API Project (ASP.NET Core MVC API)

## 2.2 WHAT FILES CAN YOU ADD?

Any file that is permissible by the respective projects listed above (*see 2.1*). For example, for an ASP.NET Core MVC project you can add a/an:

1. MVC View
2. Controller
3. Class Files
4. Images
5. CSS Files
6. JavaScript Files
7. And many, many more

## 2.3 WHY ADD NEW FILES?

You don't have to add new files, but, if you want to, you can.

Most of the time you may want to add functionality to a generated *MVC View*. **You should not do this because it will just get overwritten when you regenerate code for the same project.** Instead add a new *MVC View* and you can name it *MyNewPage.cshtml*.

## 3 ADDING CODE TO A GENERATED FILE

---

You can add your own customized code in some of the generated files. This is discussed in the *App Settings Tab* document. Please read the *App Settings Tab* document to see the list of generated files where you can add your own code to, **these files will not get overwritten even when you regenerate code for the same project.**

## 4 END-TO-END EXAMPLE ON ADDING YOUR OWN FILE AND CODE

---

In here we'll show you how to add files to the generated projects, and also add your own code to existing generated files.

### 4.1 GENERATE CODE USING ASCOREGEN 3.0 MVC PROFESSIONAL PLUS

You can generate your own Web Application using AspCoreGen 3.0 MVC Professional Plus and just follow along this tutorial. Make sure to:

1. Choose *Use Stored Procedures* under the *Generated SQL* in the *Database Settings* tab.
2. Choose *All Tables* or *Selected Tables Only* under the *Database Objects to Generate From* in the *Code Settings* tab.
3. Check the *Use Web API* under the *Web API* in the *Code Settings* tab.

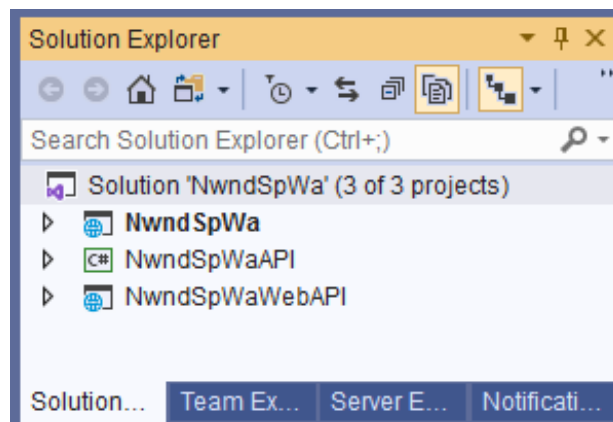
Or, you can download the sample *Generated Web Project Example* from our website:

<https://junnark.com/Products/AspCoreGen3MVC/GeneratedProjects>. Download #4, the *Stored Procedures Using Web API Sample Project*. Unzip the downloaded project and make sure to follow the instructions in the *Readme.txt* file.

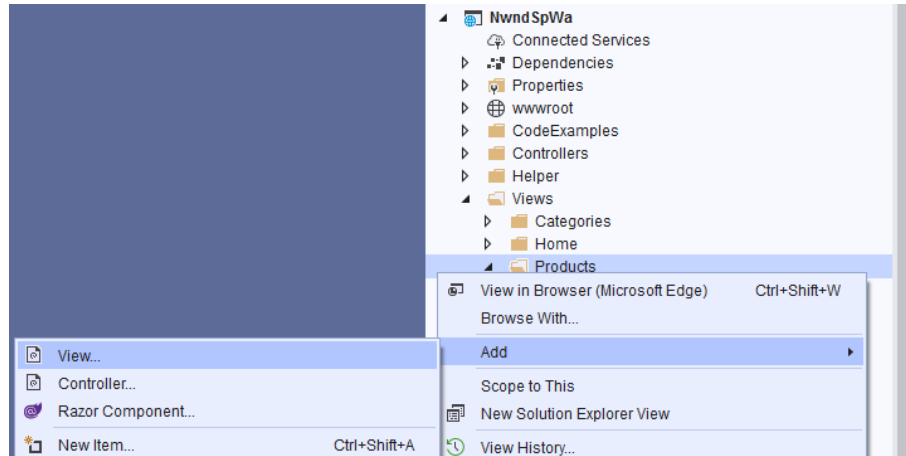
### 4.2 THE TUTORIAL

In this tutorial we're going to create a new *MVC View* that is similar to the *ListCrudRedirect.cshtml*, but we will add a functionality that shows the *Supplier Name* and *Category Name* instead of the *Supplier ID* and *Category ID* respectively. We will also remove the *UnitPrice*, *UnitsInStock*, *UnitsOnOrder*, and *ReorderLevel* columns for display.

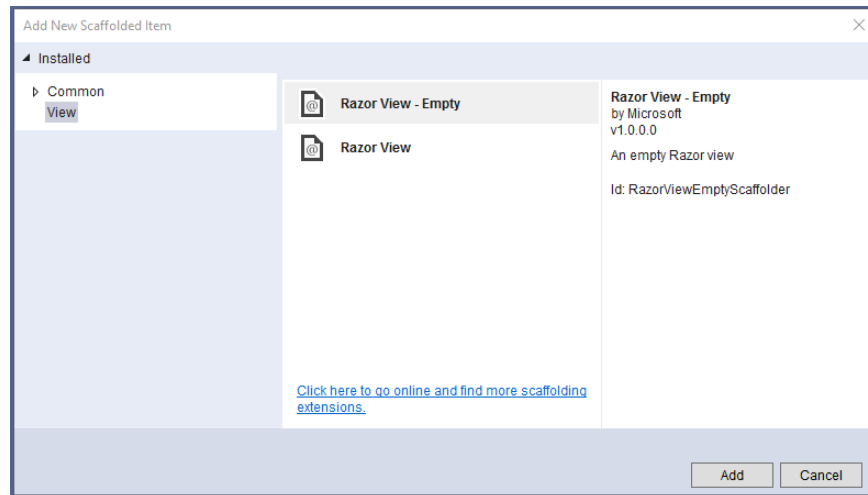
1. Open the *Generated Web Application (NwndSpWa.sln)* in *Visual Studio 2019*. This solution should have 3 projects: The *Web Application (NwndSpWa)*, the *Class Library (NwndSpWaAPI)*, and the *Web API (NwndSpWaWebAPI)* projects.



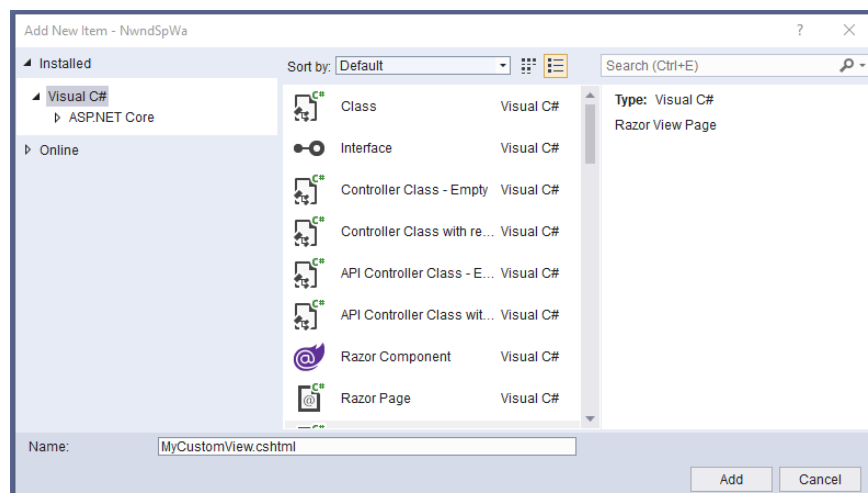
2. Add a new MVC View under the *Products* folder.



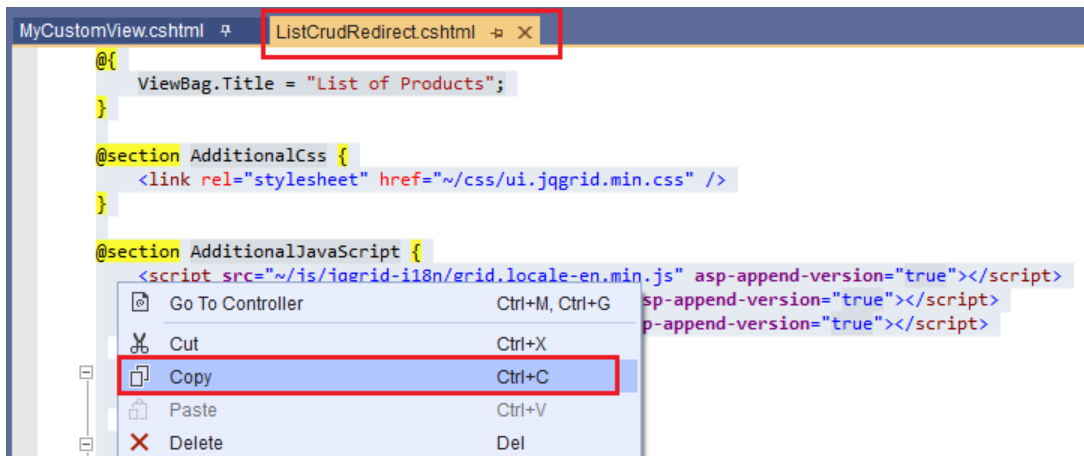
3. Choose *Razor View - Empty* and click the *Add* button.



4. Name the new MVC View: *MyCustomView.cshtml* and then click the *Add* button.



5. Delete all the commented code in the *MyCustomView.cshtml*. And then Open the *ListCrudRedirect.cshtml* under the *Products* folder and Copy all code to *MyCustomView.cshtml*.



6. Now that we've added a new file (MVC View) to the generated *Web Application Project*, we will now add code to an existing generated file. We need to add an *Action Method* for the *MyCustomView.cshtml* in the respective *ProductsController.cs*.

Again, please read the *App Settings Tab* document to see the list of generated files where you can add your own code to, **these files will not get overwritten even when you regenerate code for the same project.**

- Open the *ProductsController.cs* under the *Controllers* folder. Add an *Action Method* for the *MyCustomView.cshtml* in the respective *ProductsController.cs* as shown in red below. Also add the using statements as shown below.

```

using Microsoft.AspNetCore.Mvc;
using NwndSpWa;
using NwndSpWa.Controllers.Base;
using System.Threading.Tasks;

namespace NwndSpWa.Controllers
{
    /// <summary>
    /// This file will not be overwritten. You can put
    /// additional Products Controller code in this class.
    /// </summary>
    public class ProductsController : ProductsControllerBase
    {
        public async Task<IActionResult> MyCustomView()
        {
            // return the View
            return await Task.Run(() => View());
        }
    }
}

```

- Run the Web Application by pressing *F5* while in Visual Studio 2019. And then go to the *MyCustomView* MVC View. This page/view should look exactly like the *ListCrudRedirect.cshtml* MVC View.

List of Products

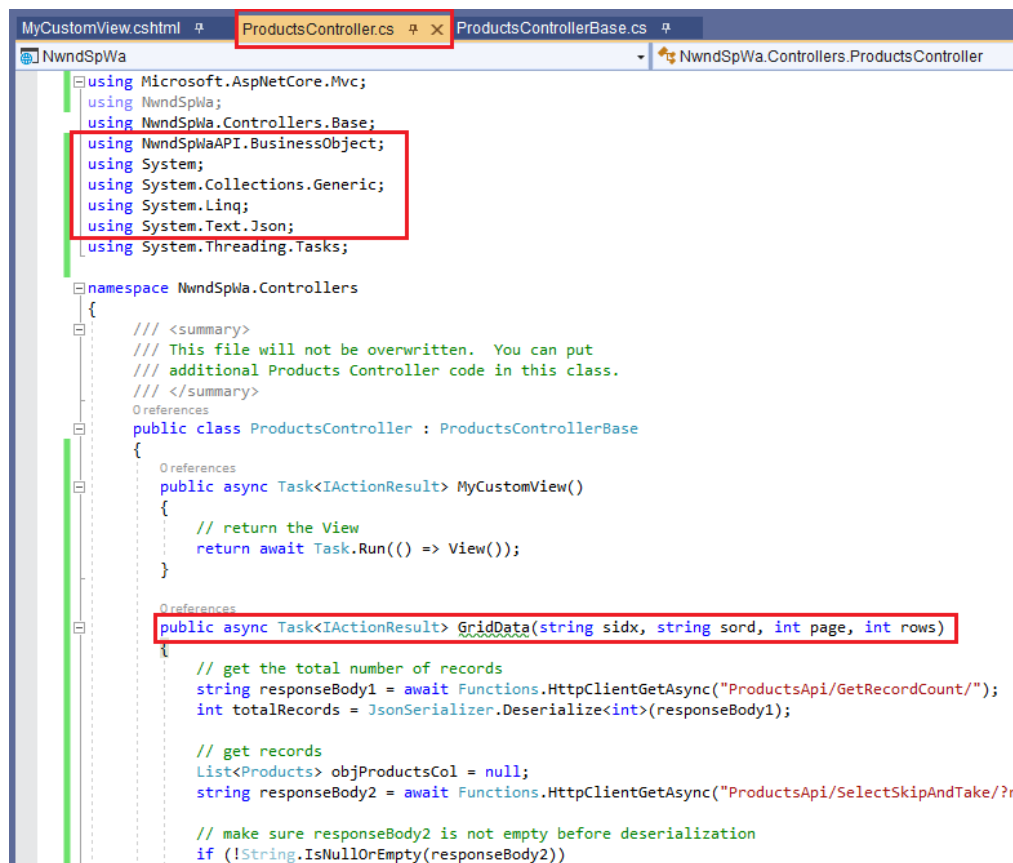
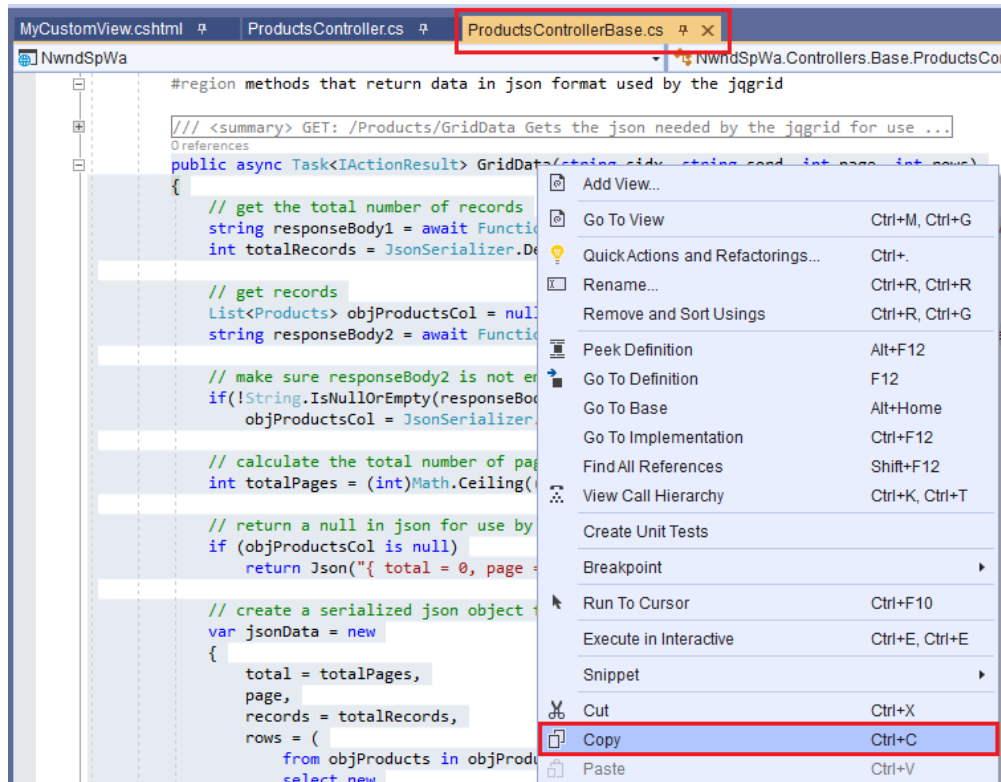
[Add New Products](#)

Product ID	Product Name	Supplier ID	Category ID	Quantity Per Unit	Unit Price	Units In Stock	Units On Order	Reorder Level	Discontinued
1	Chai	1	1	10 boxes x 20 bags	\$18.00	39	0	10	<input type="checkbox"/>
2	Chang	1	1	24 - 12 oz bottles	\$19.00	17	40	25	<input type="checkbox"/>
3	Aniseed Syrup	1	2	12 - 550 ml bottles	\$10.00	13	70	25	<input type="checkbox"/>
4	Chef Anton's Cajun S	2	2	48 - 6 oz jars	\$22.00	53	0	0	<input type="checkbox"/>
5	Chef Anton's Gumbo	2	2	36 boxes	\$21.35	0	0	0	<input checked="" type="checkbox"/>
6	Grandma's Boysenbe	3	2	12 - 8 oz jars	\$25.00	120	0	25	<input type="checkbox"/>
7	Uncle Bob's Organic	3	7	12 - 1 lb pkgs.	\$30.00	15	0	10	<input type="checkbox"/>
8	Northwoods Cranber	3	2	12 - 12 oz jars	\$40.00	6	0	0	<input type="checkbox"/>
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	\$97.00	29	0	0	<input checked="" type="checkbox"/>
10	Ikura	4	8	12 - 200 ml jars	\$31.00	31	0	0	<input type="checkbox"/>

Page 1 of 9 | View 1 - 10 of 83

- Close the browser and go back to Visual Studio 2019.

10. Open the *ProductsControllerBase.cs* (Parent/Base Class) under the *Controllers/Base* folder and then copy the *GridData* method to the *ProductsController.cs* (Child Class). Also add the using statements to the *ProductsController.cs* as shown below.





11. In the *ProductsController.cs*, change the name of the *GridData* method to *MyGridData*.



```

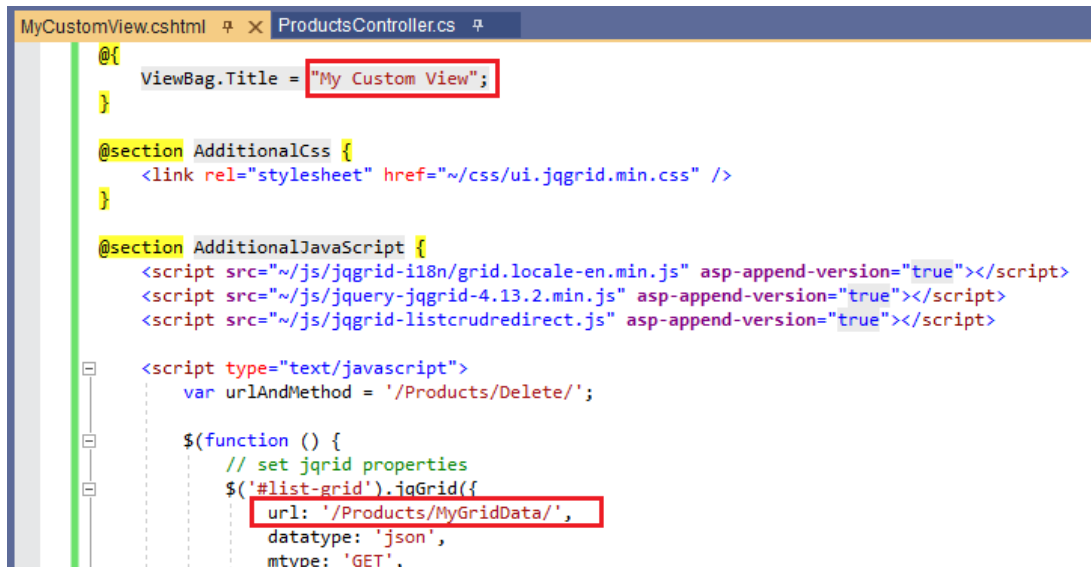
using Microsoft.AspNetCore.Mvc;
using NwndSpWa;
using NwndSpWa.Controllers.Base;
using NwndSpWaAPI.BusinessObject;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text.Json;
using System.Threading.Tasks;

namespace NwndSpWa.Controllers
{
    /// <summary>
    /// This file will not be overwritten. You can put
    /// additional Products Controller code in this class.
    /// </summary>
    [References]
    public class ProductsController : ProductsControllerBase
    {
        [References]
        public async Task<IActionResult> MyCustomView()
        {
            // return the View
            return await Task.Run(() => View());
        }

        [References]
        public async Task<IActionResult> MyGridData string sidx, string sord, int page, int rows)
        {
            // get the total number of records
            string responseBody1 = await Functions.HttpClientGetAsync("ProductsApi/GetRecordCount/");
            int totalRecords = JsonSerializer.Deserialize<int>(responseBody1);
        }
    }
}

```

12. In the *MyCustomView.cshtml*, we are going to use the new *MyGridData* method that we added on the *ProductsController.cs* as the source of the grid's data. To do this, simply change the *URL* property of *JQGrid* from *GridData* to *MyGridData* as shown below. Also change the title of the page.



```

@{
    ViewBag.Title = "My Custom View";
}

@section AdditionalCss {
    <link rel="stylesheet" href="/css/ui.jqgrid.min.css" />
}

@section AdditionalJavaScript {
    <script src="/js/jqgrid-118n/grid.locale-en.min.js" asp-append-version="true"></script>
    <script src="/js/jquery-jqgrid-4.13.2.min.js" asp-append-version="true"></script>
    <script src="/js/jqgrid-listcrudredirect.js" asp-append-version="true"></script>

    <script type="text/javascript">
        var urlAndMethod = '/Products/Delete/';

        $(function () {
            // set jqgrid properties
            $('#list-grid').jqGrid({
                url: '/Products/MyGridData/',
                datatype: 'json',
                mtype: 'GET',
            });
        });
    </script>
}

```

13. Run the *Web Application* by pressing *F5* while in Visual Studio 2019. And then go to the *MyCustomView MVC View*. This page/view should look just like the *ListCrudRedirect.cshtml MVC View* with a new page title.

https://localhost:44306/Products/MyCustomView

NwndSpWa

My Custom View

Add New Products

Product ID	Product Name	Supplier ID	Category ID	Quantity Per Unit	Unit Price	Units In Stock	Units On Order	Reorder Level	Discontinued		
1	Chai	1	1	10 boxes x 20 bags	\$18.00	39	0	10	<input type="checkbox"/>		
2	Chang	1	1	24 - 12 oz bottles	\$19.00	17	40	25	<input type="checkbox"/>		
3	Aniseed Syrup	1	2	12 - 550 ml bottles	\$10.00	13	70	25	<input type="checkbox"/>		
4	Chef Anton's Cajun S	2	2	48 - 6 oz jars	\$22.00	53	0	0	<input type="checkbox"/>		
5	Chef Anton's Gumbo	2	2	36 boxes	\$21.35	0	0	0	<input checked="" type="checkbox"/>		
6	Grandma's Boysenbe	3	2	12 - 8 oz jars	\$25.00	120	0	25	<input type="checkbox"/>		
7	Uncle Bob's Organic	3	7	12 - 1 lb pkgs.	\$30.00	15	0	10	<input type="checkbox"/>		
8	Northwoods Cranber	3	2	12 - 12 oz jars	\$40.00	6	0	0	<input type="checkbox"/>		
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	\$97.00	29	0	0	<input checked="" type="checkbox"/>		
10	Ikura	4	8	12 - 200 ml jars	\$31.00	31	0	0	<input type="checkbox"/>		

Page 1 of 9 10 View 1 - 10 of 83

14. Close the browser and go back to Visual Studio 2019.
15. Let's remove the *Unit Price*, *Units In Stock*, *Units On Order*, and *Reorder Level* from the grid. In the *MyCustomView*, delete the *Unit Price*, *Units In Stock*, *Units On Order*, and *Reorder Level* in the *colNames* and *colModel* properties of the *JQGrid*. The code should look like the one shown below after deletion.

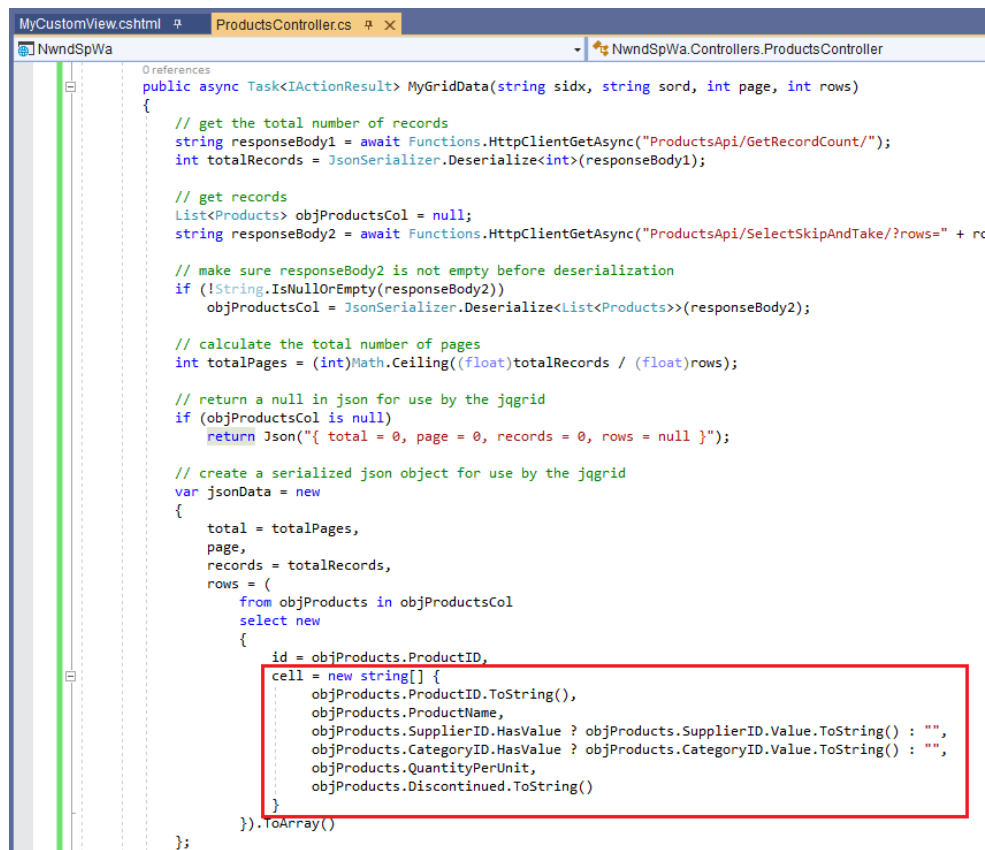
```

MyCustomView.cshtml ProductsController.cs
<script type="text/javascript">
    var urlAndMethod = '/Products/Delete/';

    $(function () {
        // set jqgrid properties
        $('#list-grid').jqGrid({
            url: '/Products/MyGridData/',
            datatype: 'json',
            mtype: 'GET',
            colNames: ['Product ID', 'Product Name', 'Supplier ID', 'Category ID', 'Quantity Per Unit', 'Discontinued', '', ''],
            colModel: [
                { name: 'ProductID', index: 'ProductID', align: 'right' },
                { name: 'ProductName', index: 'ProductName', align: 'left' },
                { name: 'SupplierID', index: 'SupplierID', align: 'right' },
                { name: 'CategoryID', index: 'CategoryID', align: 'right' },
                { name: 'QuantityPerUnit', index: 'QuantityPerUnit', align: 'left' },
                { name: 'Discontinued', index: 'Discontinued', align: 'center', formatter: 'checkbox' },
                { name: 'editoperation', index: 'editoperation', align: 'center', width: 40, sortable: false, title: false },
                { name: 'deleteoperation', index: 'deleteoperation', align: 'center', width: 40, sortable: false, title: false }
            ],
            pager: $('#list-pager'),
            rowNum: 10,

```

16. In the *ProductsController* under the *MyGridData* method, delete the lines of code that pertains to the *Unit Price*, *Units In Stock*, *Units On Order*, and *Reorder Level*. The code should look like the one shown below after deletion.



```

public async Task<IActionResult> MyGridData(string sidx, string sord, int page, int rows)
{
    // get the total number of records
    string responseBody1 = await Functions.HttpClientGetAsync("ProductsApi/GetRecordCount/");
    int totalRecords = JsonSerializer.Deserialize<int>(responseBody1);

    // get records
    List<Products> objProductsCol = null;
    string responseBody2 = await Functions.HttpClientGetAsync("ProductsApi/SelectSkipAndTake/?rows=" + rc);

    // make sure responseBody2 is not empty before deserialization
    if (!String.IsNullOrEmpty(responseBody2))
    {
        objProductsCol = JsonSerializer.Deserialize<List<Products>>(responseBody2);
    }

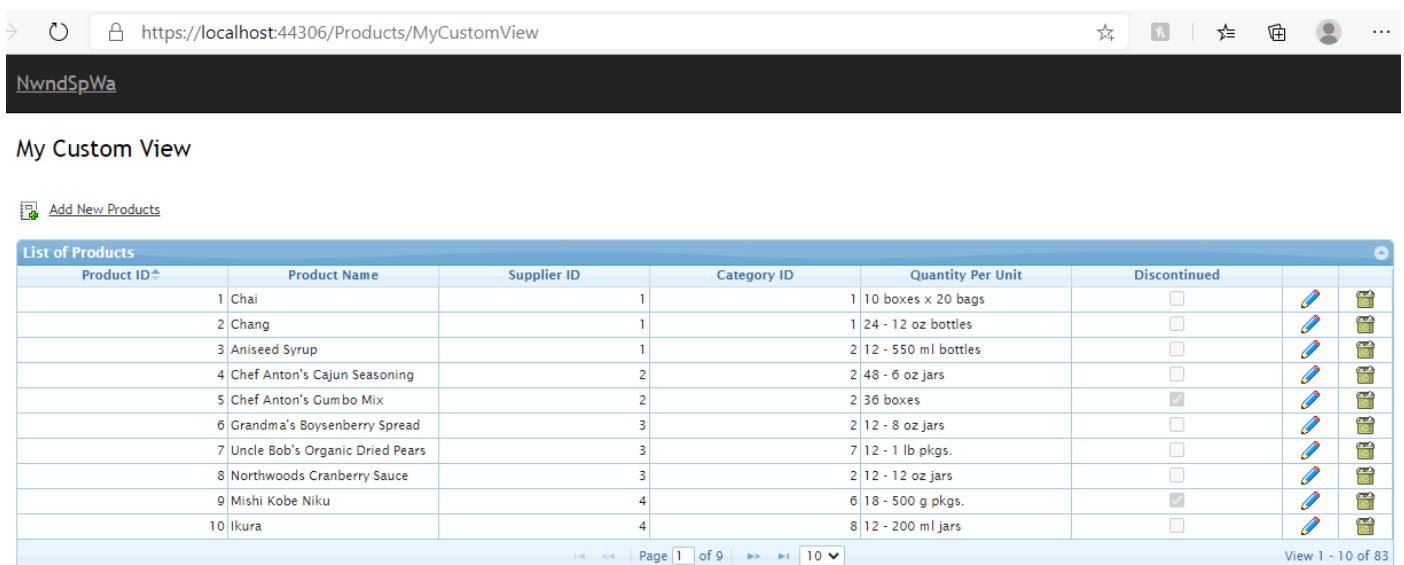
    // calculate the total number of pages
    int totalPages = (int)Math.Ceiling((float)totalRecords / (float)rows);

    // return a null in json for use by the jqgrid
    if (objProductsCol is null)
    {
        return Json("{ total = 0, page = 0, records = 0, rows = null }");
    }

    // create a serialized json object for use by the jqgrid
    var jsonData = new
    {
        total = totalPages,
        page,
        records = totalRecords,
        rows = (
            from objProducts in objProductsCol
            select new
            {
                id = objProducts.ProductID,
                cell = new string[] {
                    objProducts.ProductID.ToString(),
                    objProducts.ProductName,
                    objProducts.SupplierID.HasValue ? objProducts.SupplierID.Value.ToString() : "",
                    objProducts.CategoryID.HasValue ? objProducts.CategoryID.Value.ToString() : "",
                    objProducts.QuantityPerUnit,
                    objProducts.Discontinued.ToString()
                }
            }
        ).ToArray()
    };
}

```

17. Run the *Web Application* by pressing *F5* while in Visual Studio 2019. And then go to the *MyCustomView* MVC View. The *Unit Price*, *Units In Stock*, *Units On Order*, and *Reorder Level* should no longer be displayed on the grid.



My Custom View

[Add New Products](#)

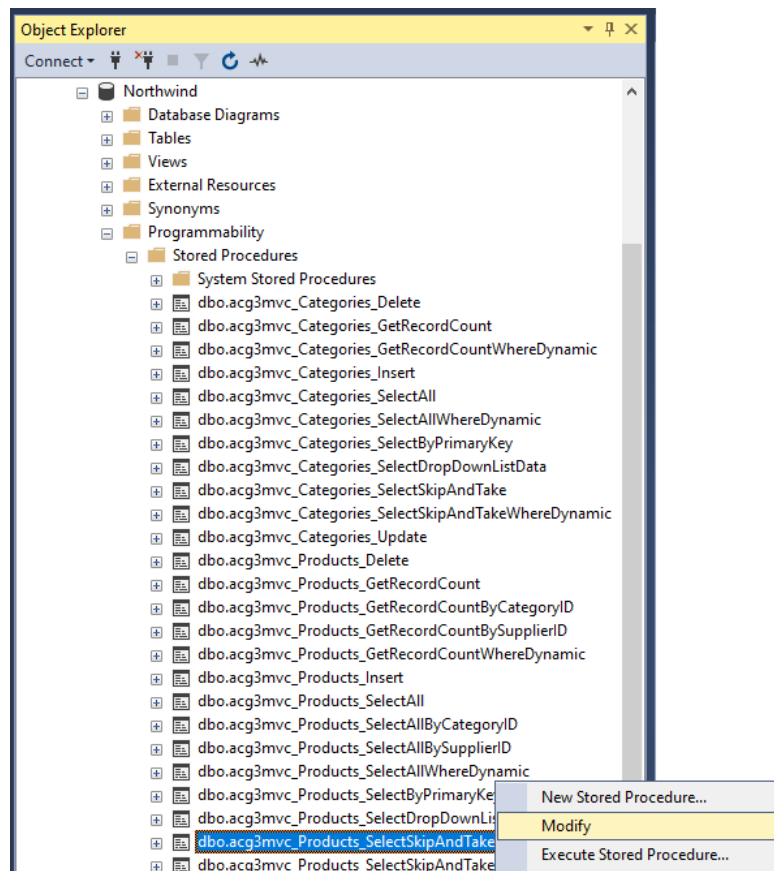
Product ID	Product Name	Supplier ID	Category ID	Quantity Per Unit	Discontinued
1	Chai	1	1	10 boxes x 20 bags	<input type="checkbox"/>
2	Chang	1	1	24 - 12 oz bottles	<input type="checkbox"/>
3	Aniseed Syrup	1	2	12 - 550 ml bottles	<input type="checkbox"/>
4	Chef Anton's Cajun Seasoning	2	2	48 - 6 oz jars	<input type="checkbox"/>
5	Chef Anton's Gumbo Mix	2	2	36 boxes	<input checked="" type="checkbox"/>
6	Grandma's Boysenberry Spread	3	2	12 - 8 oz jars	<input type="checkbox"/>
7	Uncle Bob's Organic Dried Pears	3	7	12 - 1 lb pkgs.	<input type="checkbox"/>
8	Northwoods Cranberry Sauce	3	2	12 - 12 oz jars	<input type="checkbox"/>
9	Mishi Kobe Niku	4	6	18 - 500 g pkgs.	<input checked="" type="checkbox"/>
10	Ikura	4	8	12 - 200 ml jars	<input type="checkbox"/>

Page 1 of 9 | 10 | View 1 - 10 of 83

18. Close the browser and go back to Visual Studio 2019.
19. Now we will change the display on the *Supplier ID* and *Category ID*. Instead of showing just the IDs for these foreign keys, we will show the *Company Name (Supplier)* and *Category Name (Category)* respectively. To do this, we need to:
  - a. Create a new *Stored Procedure*.
  - b. Create 2 new *Properties as Models* for *Company Name* and *Category Name*.
  - c. Create a new *Data Layer* method.
  - d. Create a new *Business Layer* method.
  - e. Create a new *Web API* method.

**Note:** There are many other ways to do this (since programming is also an art, not just science), but we'd like to walk you through the process of Adding New Code to the generated *Web Application* and Updating Existing generated code.

20. **Create a new *Stored Procedure*** named *acg3mvc\_Products\_MySelectSkipAndTake* in the *Northwind Database* using *Microsoft SQL Server Management Studio*. Go to the *Stored Procedures* folder under *Programmability* and Modify the *acg3mvc\_Products\_SelectSkipAndTake* *Stored Procedure*.



21. This will open up the *acg3mvc\_Products\_SelectSkipAndTake* Stored Procedure on a window.

```
USE [Northwind]
GO
/***** Object: StoredProcedure [dbo].[acg3mvc_Products_SelectSkipAndTake] *****/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
ALTER PROCEDURE [dbo].[acg3mvc_Products_SelectSkipAndTake]
(
    @start int,
    @numberOfRows int,
    @sortByExpression varchar(200)
)
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @numberOfRowsToSkip int = @start;

    SELECT
        [ProductID],
        [ProductName],
        [SupplierID],
        [CategoryID],
        [QuantityPerUnit],
        [UnitPrice],
        [UnitsInStock],
        [UnitsOnOrder],
        [ReorderLevel],
        [Discontinued]
    FROM [dbo].[Products]
    ORDER BY
        CASE WHEN @sortByExpression = 'ProductID' THEN [ProductID] END,
        CASE WHEN @sortByExpression = 'ProductID desc' THEN [ProductID] END DESC,
        CASE WHEN @sortByExpression = 'ProductName' THEN [ProductName] END,
        CASE WHEN @sortByExpression = 'ProductName desc' THEN [ProductName] END DESC,
```

22. Modify the *Stored Procedure*. Change the *ALTER* keyword to *CREATE*. Change the *Stored Procedure* name to *acg3mvc\_Products\_MySelectSkipAndTake*. Add *INNER JOINS* to the *Suppliers* and *Categories* tables. Remove references to the *UnitPrice*, *UnitsInStock*, *UnitsOnOrder*, and *ReorderLevel* columns.

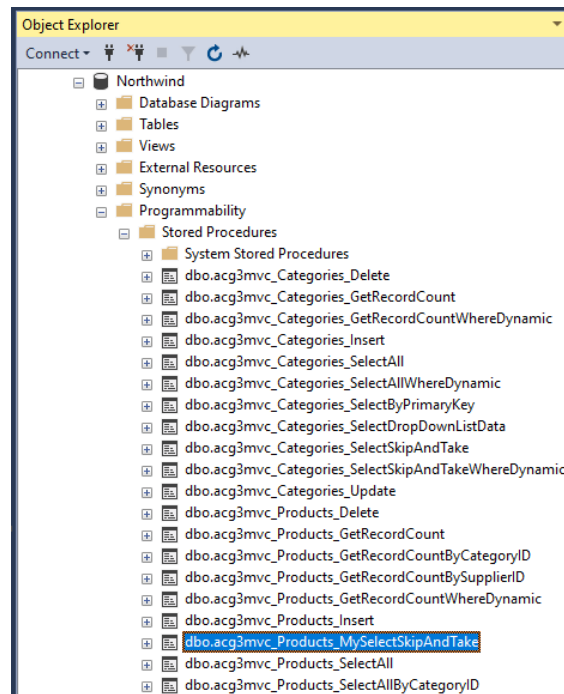
```
CREATE PROCEDURE [dbo].[acg3mvc_Products_MySelectSkipAndTake]
(
    @start int,
    @numberOfRows int,
    @sortByExpression varchar(200)
)
AS
BEGIN
    SET NOCOUNT ON;
    DECLARE @numberOfRowsToSkip int = @start;

    SELECT
        prod.[ProductID],
        prod.[ProductName],
        prod.[SupplierID],
        prod.[CategoryID],
        prod.[QuantityPerUnit],
        prod.[Discontinued],
        cat.[CategoryName],
        sup.[CompanyName]
    FROM [dbo].[Products] prod
    INNER JOIN [dbo].[Suppliers] sup
    ON prod.[SupplierID] = sup.[SupplierID]
    INNER JOIN [dbo].[Categories] cat
    ON prod.[CategoryID] = cat.[CategoryID]

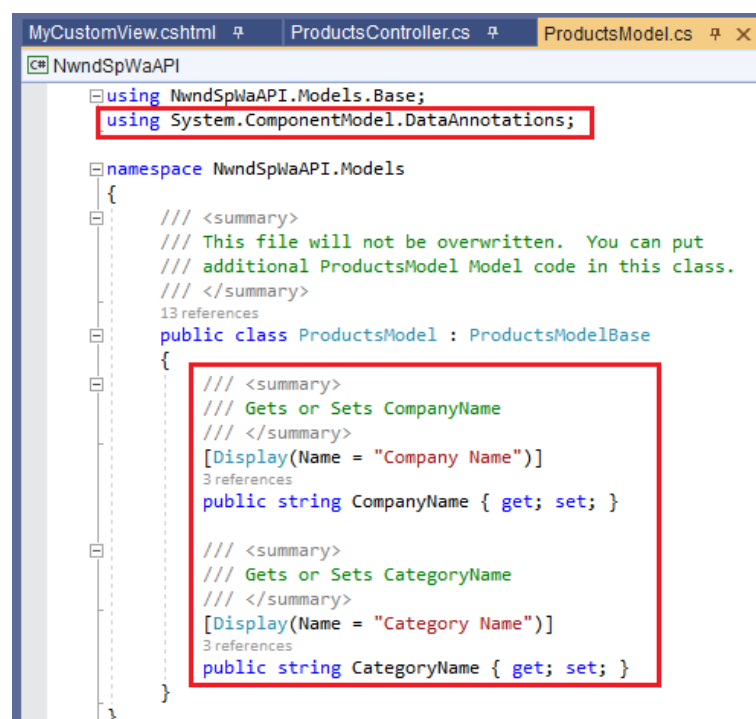
    ORDER BY
        CASE WHEN @sortByExpression = 'ProductID' THEN prod.[ProductID] END,
        CASE WHEN @sortByExpression = 'ProductID desc' THEN prod.[ProductID] END DESC,
        CASE WHEN @sortByExpression = 'ProductName' THEN prod.[ProductName] END,
        CASE WHEN @sortByExpression = 'ProductName desc' THEN prod.[ProductName] END DESC,
        CASE WHEN @sortByExpression = 'SupplierID' THEN prod.[SupplierID] END,
        CASE WHEN @sortByExpression = 'SupplierID desc' THEN prod.[SupplierID] END DESC,
        CASE WHEN @sortByExpression = 'CategoryID' THEN prod.[CategoryID] END,
        CASE WHEN @sortByExpression = 'CategoryID desc' THEN prod.[CategoryID] END DESC,
        CASE WHEN @sortByExpression = 'CompanyName' THEN sup.CompanyName END,
        CASE WHEN @sortByExpression = 'CompanyName desc' THEN sup.CompanyName END DESC,
        CASE WHEN @sortByExpression = 'CategoryName' THEN cat.CategoryName END,
        CASE WHEN @sortByExpression = 'CategoryName desc' THEN cat.CategoryName END DESC,
        CASE WHEN @sortByExpression = 'Discontinued' THEN prod.[Discontinued] END,
        CASE WHEN @sortByExpression = 'Discontinued desc' THEN prod.[Discontinued] END DESC

    OFFSET @numberOfRowsToSkip ROWS
    FETCH NEXT @numberOfRows ROWS ONLY
END
```

23. Make sure to click *Execute* in the *Microsoft SQL Server Management Studio's* menu to create the *acg3mvc\_Products\_MySelectSkipAndTake* Stored Procedure. When you refresh the Stored Procedures, the *acg3mvc\_Products\_MySelectSkipAndTake* should now be displayed.



24. Create 2 new *Properties as Models* for *CompanyName* and *CategoryName*. Open the *ProductsModel.cs* located in the *NwndSpWaAPI (Class Library Project)* under the *Models* folder. Add the *CompanyName* (*Suppliers Database Table*) and *CategoryName* (*Categories Database Table*) properties. Also add the using statement as shown below.



```
MyCustomView.cshtml  ProductsController.cs  ProductsDataLayerBase.cs  ProductsDataLa
NwndSpWaaPI  NwndSpWaaPI

/// <summary> Creates a Products object from the passed data row
4 references
private static Products CreateProductsFromDataRowShared(DataRow dr)
{
    Products objProducts = new Products();

    objProducts.ProductID = (int)dr["ProductID"];
    objProducts.ProductName = dr["ProductName"].ToString();

    if (dr["SupplierID"] != System.DBNull.Value)
        objProducts.SupplierID = (int)dr["SupplierID"];
    else
        objProducts.SupplierID = null;

    if (dr["CategoryID"] != System.DBNull.Value)
        objProducts.CategoryID = (int)dr["CategoryID"];
    else
        objProducts.CategoryID = null;

    if (dr["QuantityPerUnit"] != System.DBNull.Value)
        objProducts.QuantityPerUnit = dr["QuantityPerUnit"].ToString();
    else
        objProducts.QuantityPerUnit = null;
}
```

```

using System;
using NwndSpWAPI.DataLayer.Base;

namespace NwndSpWAPI.DataLayer
{
    /// <summary>
    /// This file will not be overwritten. You can put
    /// additional Products DataLayer code in this class
    /// </summary>

    20 references
    internal class ProductsDataLayer : ProductsDataLayerBase
    {
        // constructor
        0 references
        internal ProductsDataLayer()
        {
        }

        1 reference
        internal static async Task<List<Products>> SelectSkipAndTakeAsync(string sortByExpre

        1 reference
        internal static async Task<List<Products>> SelectSharedAsync(string storedProcName,

        1 reference
        private static Products CreateProductsFromDataRowShared(DataRow dr)...
    }

```

26. Change the name of the following methods in the *ProductsDataLayer.cs*. Also add the using statements as shown below.

- SelectSkipAndTakeAsync* to **MySelectSkipAndTakeAsync**.
- SelectSharedAsync* to **MySelectSharedAsync**.
- CreateProductsFromDataRowShared* to **MyCreateProductsFromDataRowShared**.

```

using System;
using System.Collections.Generic;
using System.Data;
using System.Data.SqlClient;
using System.Threading.Tasks;
using NwndSpWAPI.BusinessObject;
using NwndSpWAPI.DataLayer.Base;

namespace NwndSpWAPI.DataLayer
{
    /// <summary>
    /// This file will not be overwritten. You can put
    /// additional Products DataLayer code in this class
    /// </summary>

    20 references
    internal class ProductsDataLayer : ProductsDataLayerBase
    {
        // constructor
        0 references
        internal ProductsDataLayer()
        {
        }

        1 reference
        internal static async Task<List<Products>> MySelectSkipAndTakeAsync(string sortByExpre

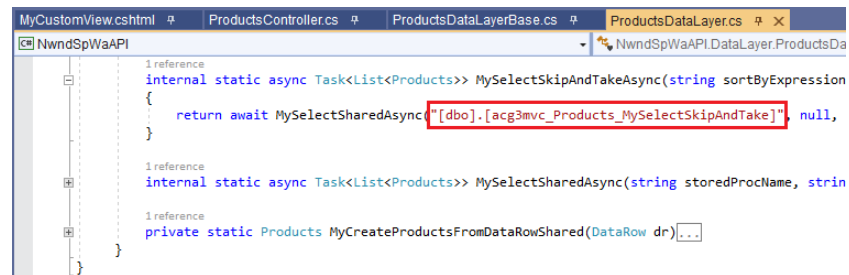
        1 reference
        internal static async Task<List<Products>> MySelectSharedAsync(string storedProcName,

        1 reference
        private static Products MyCreateProductsFromDataRowShared(DataRow dr)...
    }

```



27. Change the *Stored Procedure* name to **acg3mvc\_Products\_MySelectSkipAndTake** under the *MySelectSkipAndTakeAsync* method. This is the *Stored Procedure* that we created earlier.



```

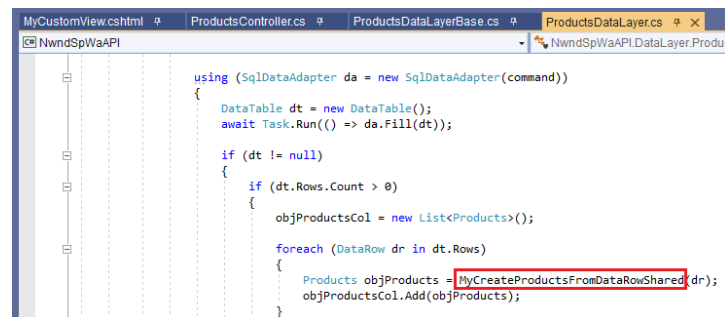
1 reference
internal static async Task<List<Products>> MySelectSkipAndTakeAsync(string sortByExpression
{
    return await MySelectSharedAsync("[dbo].[acg3mvc_Products_MySelectSkipAndTake]", null,
}

1 reference
internal static async Task<List<Products>> MySelectSharedAsync(string storedProcName, strin

1 reference
private static Products MyCreateProductsFromDataRowShared(DataRow dr)...

```

28. Change the *CreateProductsFromDataSource* method reference to **MyCreateProductsFromDataSource** under the *MySelectSharedAsync* method.



```

using (SqlDataAdapter da = new SqlDataAdapter(command))
{
    DataTable dt = new DataTable();
    await Task.Run(() => da.Fill(dt));

    if (dt != null)
    {
        if (dt.Rows.Count > 0)
        {
            objProductsCol = new List<Products>();

            foreach (DataRow dr in dt.Rows)
            {
                Products objProducts = MyCreateProductsFromDataRowShared(dr);
                objProductsCol.Add(objProducts);
            }
        }
    }
}

```

29. Add code assignments for the *CompanyName* and *CategoryName* in the **MyCreateProductsFromDataSource** method. Also, remove references to the *UnitPrice*, *UnitsInStock*, *UnitsOnOrder*, and *ReorderLevel*.



```

1 reference
private static Products MyCreateProductsFromDataRowShared(DataRow dr)
{
    Products objProducts = new Products();

    objProducts.ProductID = (int)dr["ProductID"];
    objProducts.ProductName = dr["ProductName"].ToString();

    if (dr["SupplierID"] != System.DBNull.Value)
        objProducts.SupplierID = (int)dr["SupplierID"];
    else
        objProducts.SupplierID = null;

    if (dr["CategoryID"] != System.DBNull.Value)
        objProducts.CategoryID = (int)dr["CategoryID"];
    else
        objProducts.CategoryID = null;

    if (dr["CompanyName"] != System.DBNull.Value)
        objProducts.CompanyName = dr["CompanyName"].ToString();
    else
        objProducts.CompanyName = null;

    if (dr["CategoryName"] != System.DBNull.Value)
        objProducts.CategoryName = dr["CategoryName"].ToString();
    else
        objProducts.CategoryName = null;

    if (dr["QuantityPerUnit"] != System.DBNull.Value)
        objProducts.QuantityPerUnit = dr["QuantityPerUnit"].ToString();
    else
        objProducts.QuantityPerUnit = null;

    objProducts.Discontinued = (bool)dr["Discontinued"];

    return objProducts;
}

```

30. Create a new **Business Layer** method. Open the *ProductsBase.cs* (Parent/Base Class) and the *Products.cs* (Child Class) under the *BusinessObject /Base* and *BusinessObject* folders respectively. Copy the following methods to the *Products.cs* from the *ProductsBase.cs*:

- SelectSkipAndTakeAsync* method.
- GetSortExpression* method.

This screenshot shows the *ProductsBase.cs* file in Visual Studio. The *SelectSkipAndTakeAsync* method is highlighted in blue. The method signature is `public static async Task<List<Products>> SelectSkipAndTakeAsync(int rows, int startRowIndex, string sortByExpression)`. The implementation calls `GetSortExpression` and then `ProductsDataLayer.SelectSkipAndTakeAsync`.

```

/// <summary> Selects records as a collection (List) of Products sorted by the s ...
8 references
public static async Task<List<Products>> SelectSkipAndTakeAsync(int rows, int startRowIndex, string sortByExpression)
{
    sortByExpression = GetSortExpression(sortByExpression);
    return await ProductsDataLayer.SelectSkipAndTakeAsync(sortByExpression, startRowIndex, rows);
}

```

This screenshot shows the *ProductsBase.cs* file in Visual Studio. The *GetSortExpression* method is highlighted in blue. The method signature is `private static string GetSortExpression(string sortByExpression)`. The implementation checks if the `sortByExpression` is null or empty or equals "asc", and if so, sets it to "ProductID". Otherwise, it removes "asc" from the end of the expression.

```

4 references
private static string GetSortExpression(string sortByExpression)
{
    // when no sort expression is provided, ProductID is set as the default in ascending order
    // for ascending order, "asc" is not needed, so it is removed
    if (String.IsNullOrEmpty(sortByExpression) || sortByExpression == " asc")
        sortByExpression = "ProductID";
    else if (sortByExpression.Contains(" asc"))
        sortByExpression = sortByExpression.Replace(" asc", "");
    return sortByExpression;
}

```

This screenshot shows the *Products.cs* file in Visual Studio. It contains the *Products* class, which is a partial class derived from *ProductsBase*. The *SelectSkipAndTakeAsync* and *GetSortExpression* methods are copied from *ProductsBase.cs* and are highlighted in blue. The *Products* class also includes using statements for *System* and *NwndSpWaAPI.BusinessObject.Base*.

```

using System;
using NwndSpWaAPI.BusinessObject.Base;

namespace NwndSpWaAPI.BusinessObject
{
    /// <summary>
    /// This file will not be overwritten. You can put
    /// additional Products Business Layer code in this class.
    /// </summary>
    99+ references
    public partial class Products : ProductsBase
    {
        0 references
        public static async Task<List<Products>> SelectSkipAndTakeAsync(int rows, int startRowIndex, string sortByExpression)
        {
            sortByExpression = GetSortExpression(sortByExpression);
            return await ProductsDataLayer.SelectSkipAndTakeAsync(sortByExpression, startRowIndex, rows);
        }

        1 reference
        private static string GetSortExpression(string sortByExpression)
        {
            // when no sort expression is provided, ProductID is set as the default in ascending order
            // for ascending order, "asc" is not needed, so it is removed
            if (String.IsNullOrEmpty(sortByExpression) || sortByExpression == " asc")
                sortByExpression = "ProductID";
            else if (sortByExpression.Contains(" asc"))
                sortByExpression = sortByExpression.Replace(" asc", "");
            return sortByExpression;
        }
    }
}

```

31. Change the name of the following methods in the *Products.cs*. Also add the using statements as shown below.

- SelectSkipAndTakeAsync* to **MySelectSkipAndTakeAsync**.
- GetSortExpression* to **MyGetSortExpression**.

Also, under the *MySelectSkipAndTakeAsync* method, change the *GetSortExpression* reference to **MyGetSortExpression** and the *SelectSkipAndTakeAsync* reference to **MySelectSkipAndTakeAsync**.

```

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using NwndSpWaAPI.BusinessObject.Base;
using NwndSpWaAPI.DataLayer;

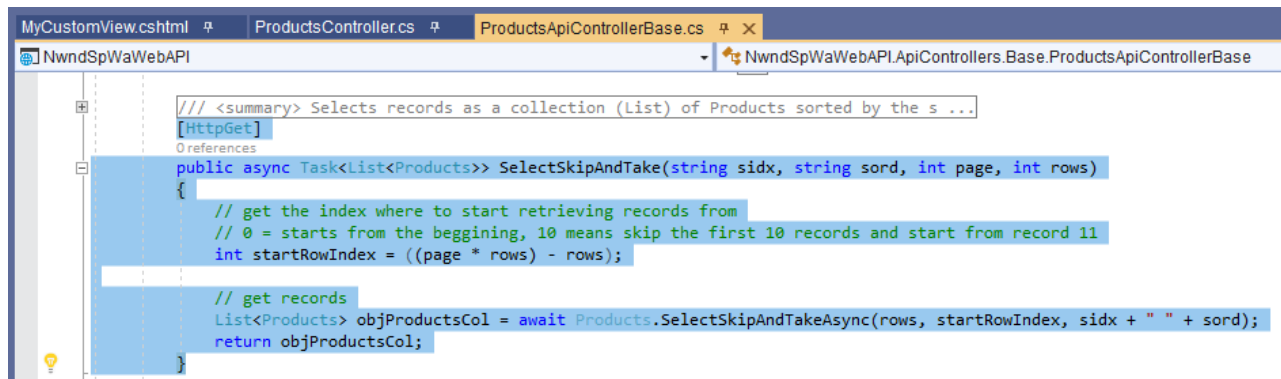
namespace NwndSpWaAPI.BusinessObject
{
    /// <summary>
    /// This file will not be overwritten. You can put
    /// additional Products Business Layer code in this class.
    /// </summary>
    public partial class Products : ProductsBase
    {
        1 reference
        public static async Task<List<Products>> MySelectSkipAndTakeAsync(int rows, int startRowIndex, string sortByExpression)
        {
            sortByExpression = MyGetSortExpression(sortByExpression);
            return await ProductsDataLayer.MySelectSkipAndTakeAsync(sortByExpression, startRowIndex, rows);
        }

        1 reference
        private static string MyGetSortExpression(string sortByExpression)
        {
            // when no sort expression is provided, ProductID is set as the default in ascending order
            // for ascending order, "asc" is not needed, so it is removed
            if (String.IsNullOrEmpty(sortByExpression) || sortByExpression == " asc")
            {
                sortByExpression = "ProductID";
            }
            else if (sortByExpression.Contains(" asc"))
            {
                sortByExpression = sortByExpression.Replace(" asc", "");
            }

            return sortByExpression;
        }
    }
}

```

32. **Create a new Web API method.** Copy the *SelectSkipAndTake* method from the *ProductsApiControllerBase.cs* (Parent/Base Class) to the *ProductsApiController.cs* (Child Class). Both the *ProductsApiControllerBase.cs* and *ProductsApiController.cs* are in the Web API Project (*NwndSpWaWebAPI*) under the *Controllers/Base* and *Controllers* folders respectively. Also add the using statements as shown below.



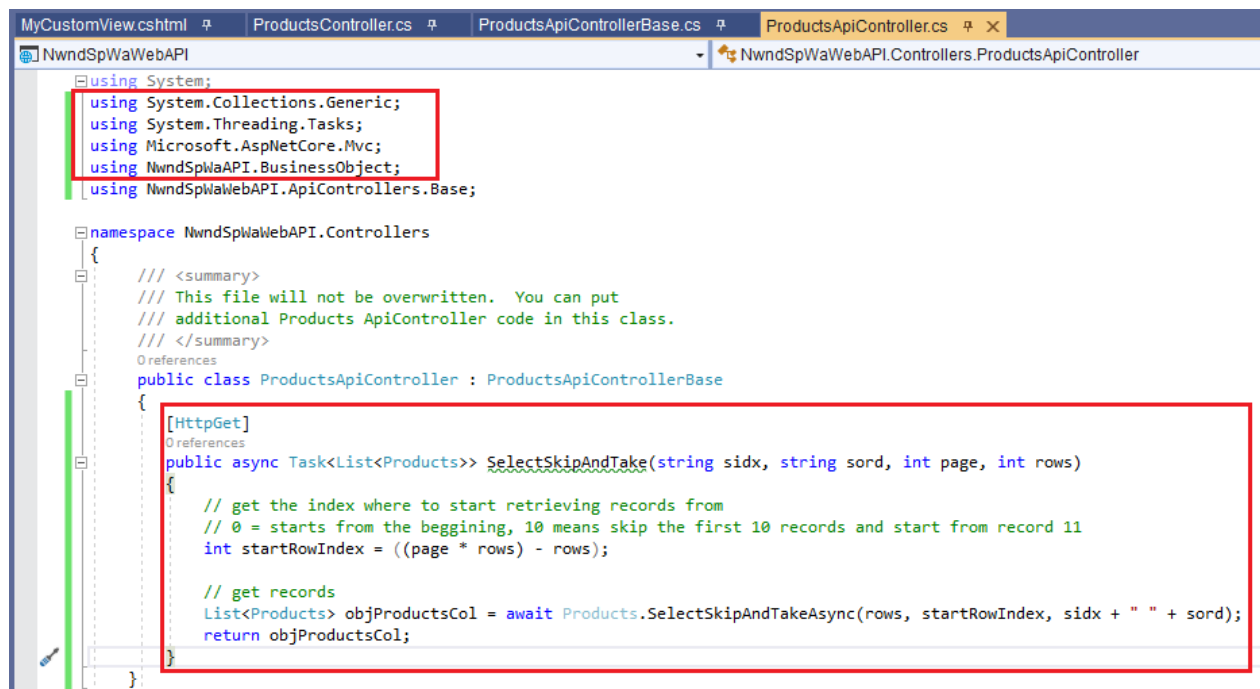
```

MyCustomView.cshtml  ProductsController.cs  ProductsApiControllerBase.cs  X
NwndSpWaWebAPI
NwndSpWaWebAPI.ApiControllers.Base.ProductsApiControllerBase

/// <summary> Selects records as a collection (List) of Products sorted by the s ...
[HttpGet]
public async Task<List<Products>> SelectSkipAndTake(string sidx, string sord, int page, int rows)
{
    // get the index where to start retrieving records from
    // 0 = starts from the beggining, 10 means skip the first 10 records and start from record 11
    int startRowIndex = ((page * rows) - rows);

    // get records
    List<Products> objProductsCol = await Products.SelectSkipAndTakeAsync(rows, startRowIndex, sidx + " " + sord);
    return objProductsCol;
}

```



```

MyCustomView.cshtml  ProductsController.cs  ProductsApiControllerBase.cs  ProductsApiController.cs  X
NwndSpWaWebAPI
NwndSpWaWebAPI.Controllers.ProductsApiController

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using NwndSpWaAPI.BusinessObject;
using NwndSpWaWebAPI.ApiControllers.Base;

namespace NwndSpWaWebAPI.Controllers
{
    /// <summary>
    /// This file will not be overwritten. You can put
    /// additional Products ApiController code in this class.
    /// </summary>
    public class ProductsApiController : ProductsApiControllerBase
    {
        [HttpGet]
        public async Task<List<Products>> SelectSkipAndTake(string sidx, string sord, int page, int rows)
        {
            // get the index where to start retrieving records from
            // 0 = starts from the beggining, 10 means skip the first 10 records and start from record 11
            int startRowIndex = ((page * rows) - rows);

            // get records
            List<Products> objProductsCol = await Products.SelectSkipAndTakeAsync(rows, startRowIndex, sidx + " " + sord);
            return objProductsCol;
        }
    }
}

```

33. In the *ProductsApiController*, change the *SelectSkipAndTake* Web API method name to **MySelectSkipAndTake**. Also change the *SelectSkipAndTakeAsync* (Business Object) name reference to **MySelectSkipAndTakeAsync** as shown below.

```

using System;
using System.Collections.Generic;
using System.Threading.Tasks;
using Microsoft.AspNetCore.Mvc;
using NwndSpWaAPI.BusinessObject;
using NwndSpWaWebAPI.ApiControllers.Base;

namespace NwndSpWaWebAPI.Controllers
{
    /// <summary>
    /// This file will not be overwritten. You can put
    /// additional Products ApiController code in this class.
    /// </summary>
    public class ProductsApiController : ProductsApiControllerBase
    {
        [HttpGet]
        public async Task<List<Products>> MySelectSkipAndTake(string sidx, string sord, int page, int rows)
        {
            // get the index where to start retrieving records from
            // 0 = starts from the beginning, 10 means skip the first 10 records and start from record 11
            int startRowIndex = ((page * rows) - rows);

            // get records
            List<Products> objProductsCol = await Products.MySelectSkipAndTakeAsync(rows, startRowIndex, sidx + " " + sord);
            return objProductsCol;
        }
    }
}

```

34. Now that all the code plumbing is done, we just need to display the *CompanyName* and *CategoryName* as we originally intended. First we need to reference the *MySelectSkipAndTake* Web API, and then add code that will display the *CompanyName* and *CategoryName* under the *ProductsController* as shown below.

```

public async Task<IActionResult> MyGridData(string sidx, string sord, int page, int rows)
{
    // get the total number of records
    string responseBody1 = await Functions.HttpClientGetAsync("ProductsApi/GetRecordCount/");
    int totalRecords = JsonSerializer.Deserialize<int>(responseBody1);

    // get records
    List<Products> objProductsCol = null;
    string responseBody2 = await Functions.HttpClientGetAsync("ProductsApi/MySelectSkipAndTake?rows=" + rows + "&page=" + page + "&sidx=" + sidx + "&sord=" + sord);

    // make sure responseBody2 is not empty before deserialization
    if (!String.IsNullOrEmpty(responseBody2))
    {
        objProductsCol = JsonSerializer.Deserialize<List<Products>>(responseBody2);
    }

    // calculate the total number of pages
    int totalPages = (int)Math.Ceiling((float)totalRecords / (float)rows);

    // return a null in json for use by the jqgrid
    if (objProductsCol is null)
    {
        return Json("{ total = 0, page = 0, records = 0, rows = null }");
    }

    // create a serialized json object for use by the jqgrid
    var jsonData = new
    {
        total = totalPages,
        page,
        records = totalRecords,
        rows = (
            from objProducts in objProductsCol
            select new
            {
                id = objProducts.ProductID,
                cell = new string[] {
                    objProducts.ProductID.ToString(),
                    objProducts.ProductName,
                    objProducts.SupplierID.HasValue ? objProducts.CompanyName + " (" + objProducts.SupplierID.Value.ToString() + ")" : "",
                    objProducts.CategoryID.HasValue ? objProducts.CategoryName + " (" + objProducts.CategoryID.Value.ToString() + ")" : "",
                    objProducts.QuantityPerUnit,
                    objProducts.Discontinued.ToString()
                }
            }
        ).ToArray()
    };
}

```

35. In the *MyCustomView.cshtml* MVC View, change the *colNames* to *Supplier* and *Category* respectively. Also, remove the *SupplierID* and *CategoryID* and replace with *CompanyName* and *CategoryName* respectively as shown below.



```

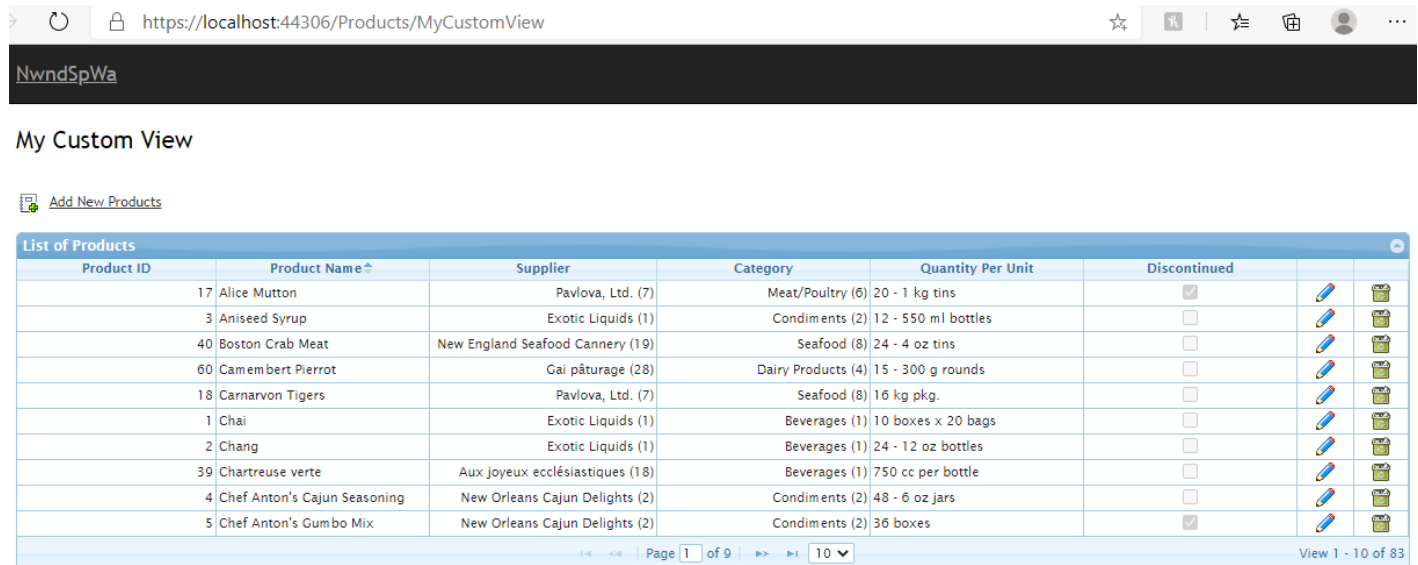
<script type="text/javascript">
    var urlAndMethod = '/Products/Delete/';

    $(function () {
        // set jqgrid properties
        $('#list-grid').jqGrid({
            url: '/Products/MyGridData/',
            datatype: 'json',
            mtype: 'GET',
            colNames: ['Product ID', 'Product Name', 'Supplier', 'Category', 'Quantity Per Unit', 'Discontinued', '', ''],
            colModel: [
                { name: 'ProductID', index: 'ProductID', align: 'right' },
                { name: 'ProductName', index: 'ProductName', align: 'left' },
                { name: 'CompanyName', index: 'CompanyName', align: 'right' },
                { name: 'CategoryName', index: 'CategoryName', align: 'right' },
                { name: 'QuantityPerUnit', index: 'QuantityPerUnit', align: 'left' },
                { name: 'Discontinued', index: 'Discontinued', align: 'center', formatter: 'checkbox' },
                { name: 'editoperation', index: 'editoperation', align: 'center', width: 40, sortable: false, title: false },
                { name: 'deleteoperation', index: 'deleteoperation', align: 'center', width: 40, sortable: false, title: false }
            ]
        });
    });

```

36. Run the *Web Application* by pressing *F5* while in Visual Studio 2019. And then go to the *MyCustomView* MVC View.

This finished MVC View no longer shows the *UnitPrice*, *UnitsInStock*, *UnitsOnOrder*, and *ReorderLevel* columns. It also shows the *CompanyName* (*Supplier*) and *CategoryName* (*Category*) with the *SupplierID* and *CategoryID* in parenthesis instead of just showing the *SupplierID* and *CategoryID* respectively. The *CompanyName* (*Supplier*) and *CategoryName* (*Category*) are also sortable.



My Custom View

Add New Products

Product ID	Product Name	Supplier	Category	Quantity Per Unit	Discontinued
17	Alice Mutton	Pavlova, Ltd. (7)	Meat/Poultry (6)	20 - 1 kg tins	<input checked="" type="checkbox"/>
3	Aniseed Syrup	Exotic Liquids (1)	Condiments (2)	12 - 550 ml bottles	<input type="checkbox"/>
40	Boston Crab Meat	New England Seafood Cannery (19)	Seafood (8)	24 - 4 oz tins	<input type="checkbox"/>
60	Camembert Pierrot	Gai pâturage (28)	Dairy Products (4)	15 - 300 g rounds	<input type="checkbox"/>
18	Carnarvon Tigers	Pavlova, Ltd. (7)	Seafood (8)	16 kg pkg.	<input type="checkbox"/>
1	Chai	Exotic Liquids (1)	Beverages (1)	10 boxes x 20 bags	<input type="checkbox"/>
2	Chang	Exotic Liquids (1)	Beverages (1)	24 - 12 oz bottles	<input type="checkbox"/>
39	Chartreuse verte	Aux joyeux ecclésiastiques (18)	Beverages (1)	750 cc per bottle	<input type="checkbox"/>
4	Chef Anton's Cajun Seasoning	New Orleans Cajun Delights (2)	Condiments (2)	48 - 6 oz jars	<input type="checkbox"/>
5	Chef Anton's Gumbo Mix	New Orleans Cajun Delights (2)	Condiments (2)	36 boxes	<input checked="" type="checkbox"/>

Page 1 of 9 View 1 - 10 of 83

You can read end-to-end tutorials on more subjects on using AspCoreGen 3.0 MVC Professional Plus that came with your purchase. These tutorials are available to customers and are included in a link on your invoice when you purchase AspCoreGen 3.0 MVC Professional.

**Note:** Some features shown here are not available in the Express Edition. The code in this tutorial is available for download for paying customers only, please email us at Software Support for more information.

End of tutorial.