



Essential Studio 2013 Volume 4 - v.11.4.0.26

## **Essential Grouping**



# Contents

<b>1</b>	<b>Overview</b>	<b>4</b>
1.1	Introduction to Essential Grouping .....	4
1.2	Prerequisites and Compatibility .....	6
1.3	Documentation .....	7
<b>2</b>	<b>Installation and Deployment</b>	<b>9</b>
2.1	Installation.....	9
2.2	Where to Find Samples? .....	9
2.3	Deployment Requirements .....	15
<b>3</b>	<b>Getting Started</b>	<b>16</b>
3.1	Creating Platform Application.....	16
3.2	Deploying Essential Grouping .....	18
3.2.1	Windows / WPF .....	19
3.2.2	ASP.NET .....	20
<b>4</b>	<b>Concepts and Features</b>	<b>22</b>
4.1	Data Binding .....	22
4.1.1	Creating an ArrayList of Objects .....	22
4.1.2	Setting a Datasource In the Grouping Engine.....	27
4.1.3	Iterating Through the Data .....	28
4.2	Using Grouping.....	30
4.2.1	Grouping a Table.....	30
4.2.1.1	The Grouping.TableDescriptor Class .....	32
4.2.2	Accessing a Particular Group.....	33
4.2.3	Adding a Summary .....	37
4.2.4	Retrieving Summary Values for a Particular Group .....	38
4.3	Data Manipulation.....	39
4.3.1	Filters .....	40
4.3.2	Expressions .....	45
4.3.3	Sorting .....	48
4.3.4	Custom Sorting.....	51
4.4	Algebra Supported In Expressions / Filters .....	56

<b>5</b>	<b>Frequently Asked Questions</b>	<b>59</b>
5.1	How to Access the Value of a Record Or Field? .....	59
5.2	How to Add Custom Calculations to Expression Fields? .....	60
5.3	How to Add Expression Fields? .....	62
5.4	How to Add Summary Items?.....	63
5.5	How to Bind a Datasource to the Grouping Engine? .....	63
5.6	How to Clear a Filter? .....	64
5.7	How to Clear a Grouping? .....	65
5.8	How to Clear a Sort? .....	66
5.9	How to Filter a Collection?.....	66
5.10	How to Group a Collection?.....	67
5.11	How to Retrieve Summary Item Values? .....	67
5.12	How to Sort a Collection? .....	69

## 1 Overview

---

This section covers information on Essential Grouping, its key features, prerequisites to use the control, its compatibility with various OS and browsers, and finally the documentation details complimentary with the product. It comprises the following subsections:

### 1.1 Introduction to Essential Grouping

Essential Grouping is a 100% Native .NET library that provides you with support for managing and manipulating tabular information without dependencies on any particular UI component. Our grouping framework can be used in any .NET environment, including C#, VB.NET, and managed C++.

Syncfusion Essential Grouping is a data technology that allows you to easily access, manipulate, and display your data in a variety of configurations. Your data source can be any IList object whose items have public properties. You can easily sort the items on one or several of these public properties. You can display and retrieve items based on the grouping that is produced through these sorts, you can include caption information and / or summary information on these groups; you can impose filters on the items, retrieving only items that specify your filter conditions and you can also add expression properties to display calculated values depending upon other properties in the item.

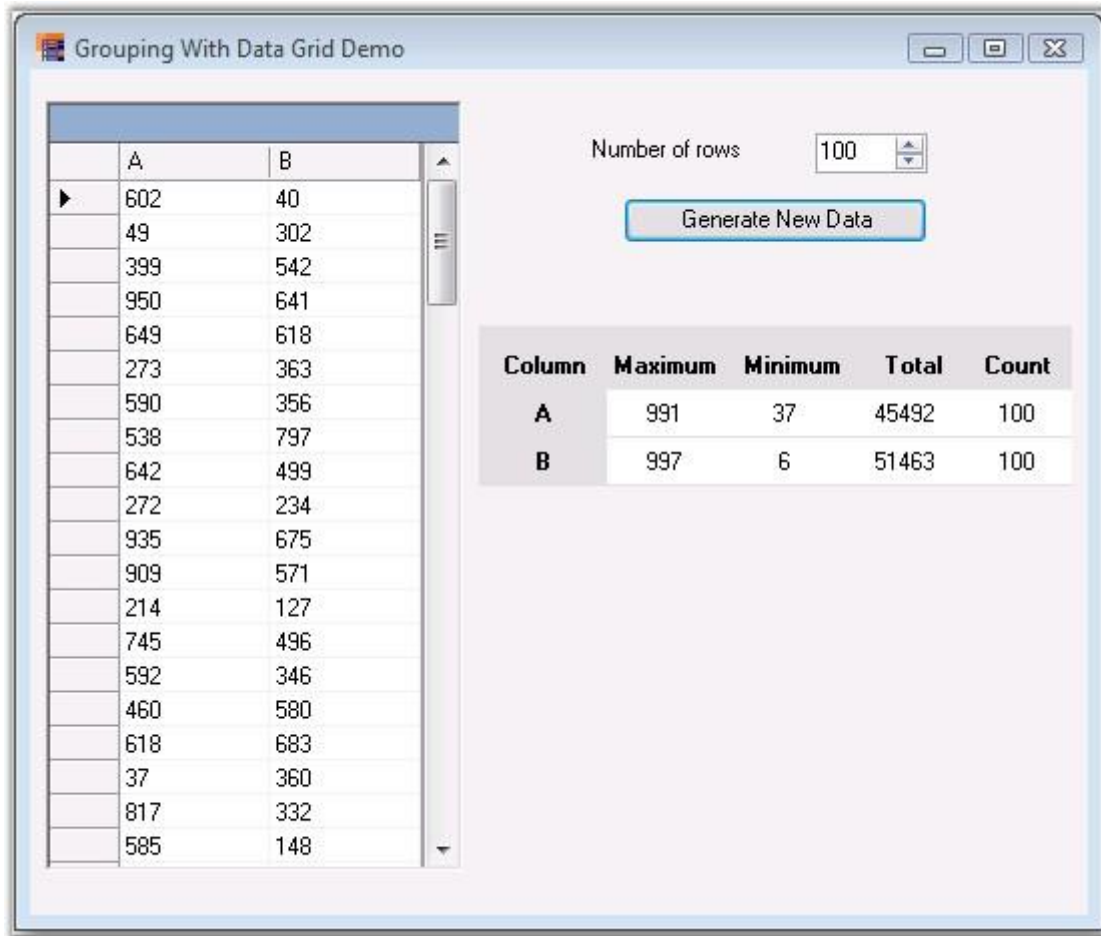


Figure 1: Grouping in Data Grid

## Key Features

Some of the key features of Essential Grouping are listed below:

- Grouping supports data presentation techniques like sorting, grouping, adding caption and summary information for the groups.
- Grouping also supports nested tables and hierarchies in the form of related tables.
- The grouping technology uses balanced binary trees as the core data structure instead of arrays. Binary trees have this advantage whereby parent branches can cache information about their children. This allows position information and summary information to be cached in parent branches facilitating quick inserts of new records honoring any sort of criteria that is applied. Inserting, removing, and moving of records takes \ only  $\text{Log}_2(n)$  operations. With linear lookup structures such as an ArrayList, each of these operations would take  $O(n)$  operations.
- Expressions can be any well-formed algebraic combination of property (column) names enclosed with brackets ([ ]), numerical constants and literals, and the algebraic and logical operators.

- Grouping is a recursive process whereby a data source may be grouped several times. This leads to the recursive situation of groups having sub-groups and so on.

### User Guide Organization




The product comes with numerous samples as well as an extensive documentation to guide you. This User Guide provides detailed information on the features and functionalities of Essential Grouping. It is organized into the following sections:

- **Overview**-This section gives a brief introduction to our product and its key features.
- **Installation and Deployment**-This section elaborates on the install location of the samples, license etc.
- **Getting Started**-This section guides you on getting started with various platform application, controls etc.
- **Concepts and Features**-The features of Essential Grouping is illustrated with use case scenarios, code examples and screen shots under this section.
- **Frequently Asked Questions**-This section illustrates the solutions for various task-based queries about Grouping.

### Document Conventions

The conventions listed below will help you to quickly identify the important sections of information, while using the content:

Table 1: Document Conventions

Convention	Icon	Description
Note	 <b>Note:</b>	Represents important information
Example	<b>Example</b>	Represents an example
Tip		Represents useful hints that will help you in using the controls/features
Additional Information		Represents additional information on the topic

## 1.2 Prerequisites and Compatibility

This section covers the requirements mandatory for using Essential Grouping. It also lists operating systems and browsers compatible with the product.

### Prerequisites

The prerequisites details are listed below:

Table 2: Prerequisites

Development Environments	<ul style="list-style-type: none"><li>• Visual Studio 2012 (Ultimate, Premium, Professional, and Express)</li><li>• Visual Studio 2010 (Ultimate, Premium, Professional, and Express)</li><li>• Visual Studio 2008 (Team System, Professional, Standard &amp; Express)</li><li>• Visual Studio 2005 (Professional, Standard &amp; Express)</li></ul>
.NET Framework versions	<ul style="list-style-type: none"><li>• .NET 4.5</li><li>• .NET 4.0</li><li>• .NET 3.5 SP1</li><li>• .NET 2.0</li></ul>

### Compatibility

The compatibility details are listed below:

Table 3: Compatibility

Operating Systems	<ul style="list-style-type: none"><li>• Windows 8 (32 bit and 64 bit)</li><li>• Windows Server 2012 (32 bit and 64 bit)</li><li>• Windows Server 2008 (32 bit and 64 bit)</li><li>• Windows 7 (32 bit and 64 bit)</li><li>• Windows Vista (32 bit and 64 bit)</li><li>• Windows XP</li><li>• Windows 2003</li></ul>
-------------------	---

## 1.3 Documentation

Syncfusion provides the following documentation segments to provide all necessary information for using Essential Grouping in different platform applications in an efficient manner.

Table 4: Documentation

Type of documentation	Location
-----------------------	----------

Readme	<p>1. <b>For Windows and BackOffice:</b> [drive:]\Program Files\Syncfusion\Essential Studio\<b>x.x.x.x</b>\Infrastructure\Data\Release Notes\readme.htm</p> <p>2. <b>For other platforms:</b> [drive:]\Program Files\Syncfusion\Essential Studio\<b>x.x.x.x</b>\Infrastructure\Data\&lt;<b>asp/WPF</b>&gt; Release Notes\readme.htm</p>
Release Notes	<p>1. <b>For Windows and BackOffice:</b> [drive:]\Program Files\Syncfusion\Essential Studio\<b>x.x.x.x</b>\Infrastructure\Data\Release Notes\Release Notes.htm</p> <p>2. <b>For other platforms:</b> [drive:]\Program Files\Syncfusion\Essential Studio\<b>x.x.x.x</b>\Infrastructure\Data\&lt;<b>asp/WPF</b>&gt; Release Notes\Release Notes.htm</p>
User Guide (this document)	<p><b>Online</b></p> <p><a href="http://help.syncfusion.com/resources">http://help.syncfusion.com/resources</a> (Navigate to the Grouping User Guide.)</p> <p> <b>Note: Click Download as PDF to access a PDF version.</b></p> <p><b>Installed Documentation</b></p> <p>Dashboard -&gt; Documentation -&gt; Installed Documentation.</p>
Class Reference	<p><b>Online</b></p> <p><a href="http://help.syncfusion.com/resources">http://help.syncfusion.com/resources</a> (Navigate to the Reporting User Guide. Select <i>Grouping</i>, and then click the Class Reference link found in the upper right section of the page.)</p> <p><b>Installed Documentation</b></p> <p>Dashboard -&gt; Documentation -&gt; Installed Documentation.</p>



## 2 Installation and Deployment

---

This section covers information on the install location, samples, licensing, patches update and update of the recent version of Essential Studio. It comprises the following subsections:

### 2.1 Installation

For step-by-step installation procedure for the installation of Essential Studio, refer to the **Installation** topic under **Installation and Deployment** in the **Common UG**.

#### See Also

For licensing, patches and information on adding or removing selective components refer the following topics in Common UG under **Installation and Deployment**.

- Licensing
- Patches
- Add / Remove Components

### 2.2 Where to Find Samples?

This section provides the location of the installed samples and describes the procedure to run the samples in the sample browser and online. It also lists the location of utilities, assemblies, and source code.

#### Sample Installation Location

Sample install locations for different platforms are listed below:

**Windows Forms Samples – The Grouping Windows Forms samples are installed in the following location:**

- [Install Location]:\...\Syncfusion\Essential Studio\[Version Number]\Windows\Grouping.Windows\Samples\2.0
- ASP.NET Samples – The Grouping Web samples are installed in the following location:
- [Install Location]:\...\Syncfusion\Essential Studio\[Version Number]\Web\Grouping.Web\Samples\3.5

#### Viewing Samples

To view the samples:

1. Click **Start → All Programs → Syncfusion → Essential Studio <version number> → Dashboard.**

The UI Edition samples are displayed by default.

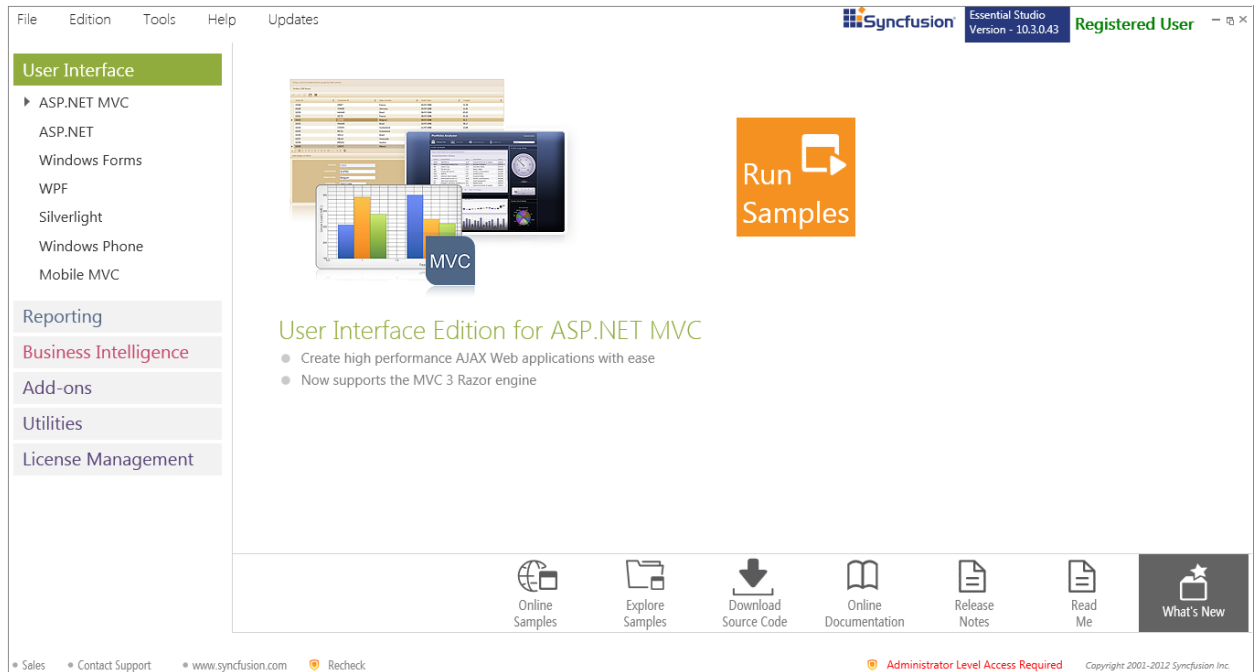


Figure 2: Syncfusion Essential Studio Dashboard

2. Select **Reporting** Edition.

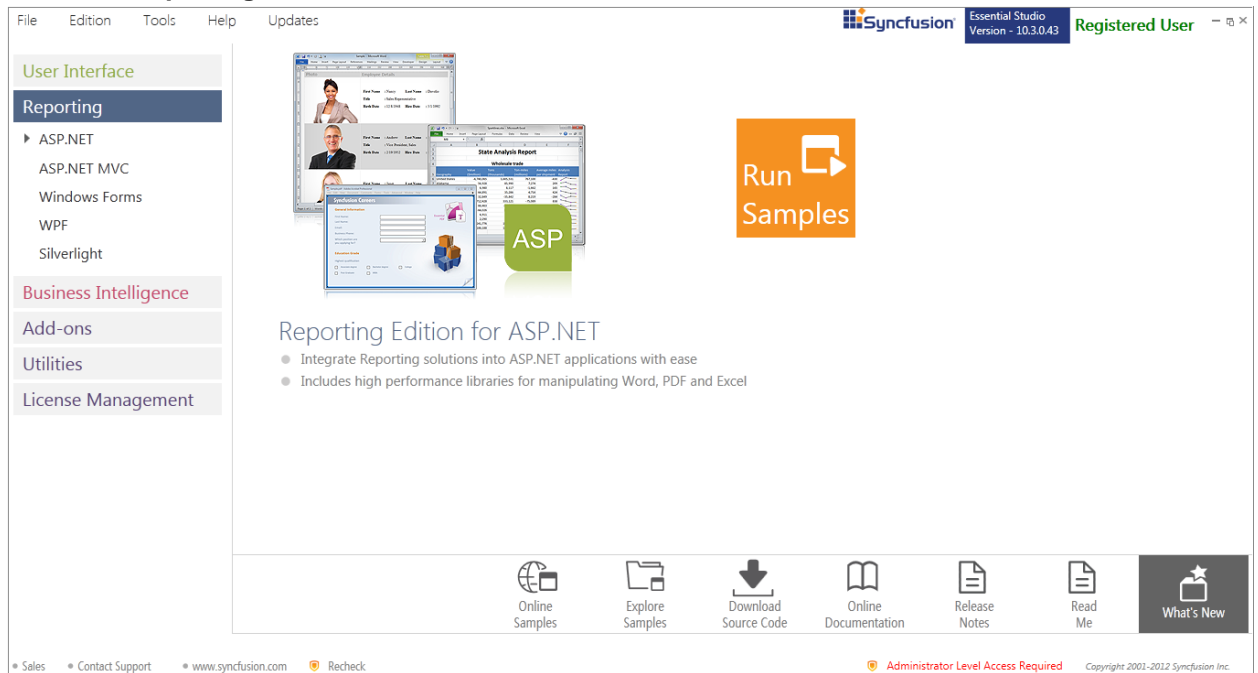


Figure 3: Syncfusion Essential Studio Dashboard

The steps to view the Grouping samples in various platforms are discussed below:

## Windows

1. In the Dashboard window, click **Run Samples** for **Windows Forms** under **Reporting Edition** panel. The **Windows Forms** Sample Browser window is displayed.

**Note:** You can view the samples in any of the following three ways:

- **Run Samples** – Click to view the locally installed samples.
- **Online Samples** – Click to view online samples.
- **Explore Samples** – Explore Windows Forms samples on disk.

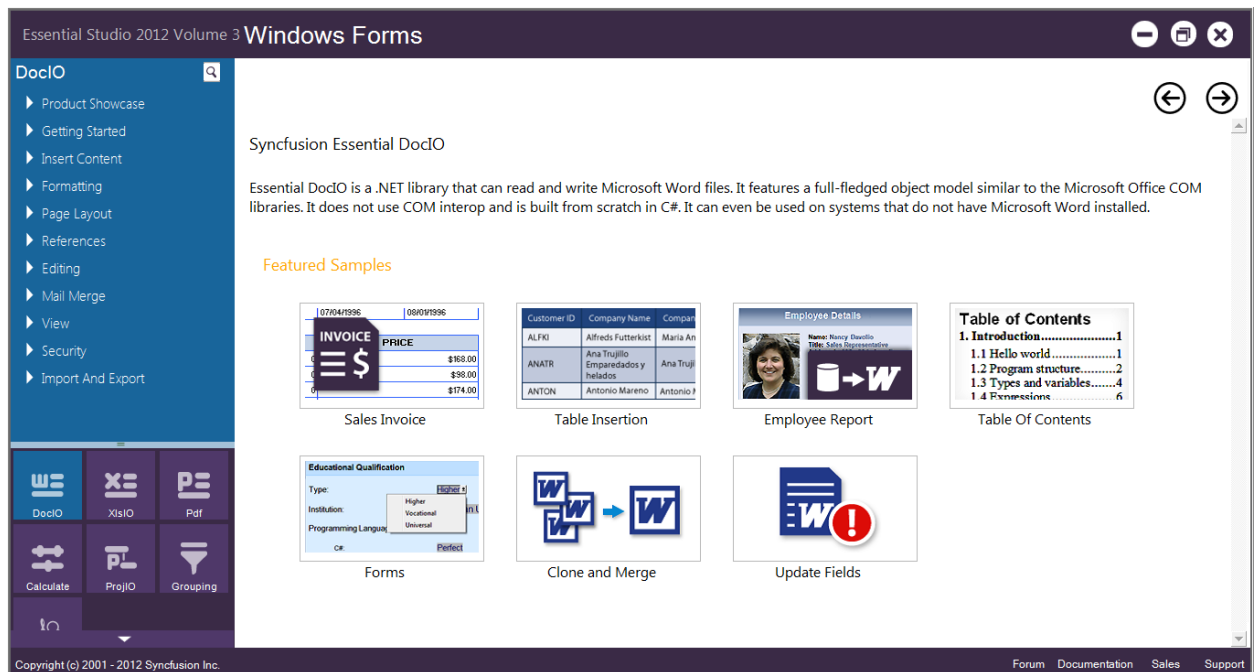


Figure 4: Windows Forms Sample Browser

2. Click **Grouping** from the bottom-left pane. The Grouping samples are displayed.

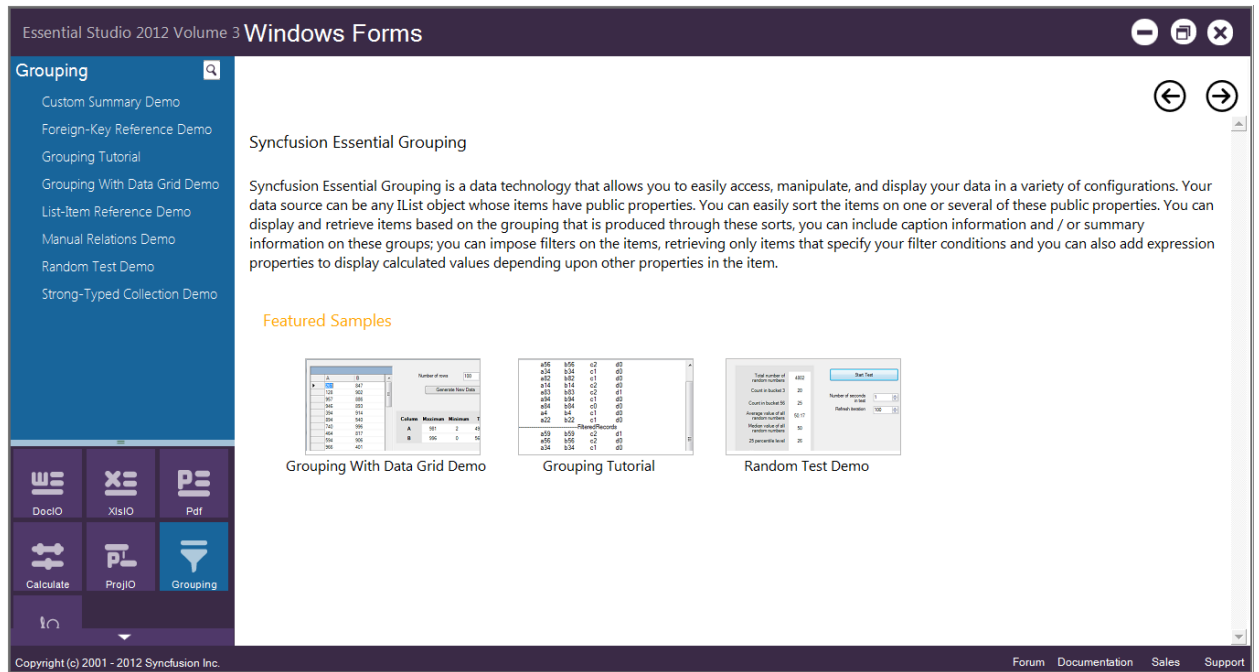


Figure 5: Grouping samples displayed in the Windows Forms Sample Browser

3. Select any sample and browse through the features.

## ASP.NET

- 1 In the Dashboard window, click **Run Samples for ASP.NET** under **Reporting Edition panel**. The ASP.NET Sample Browser window is displayed.

**Note:** You can view the samples in any of the three ways displayed:

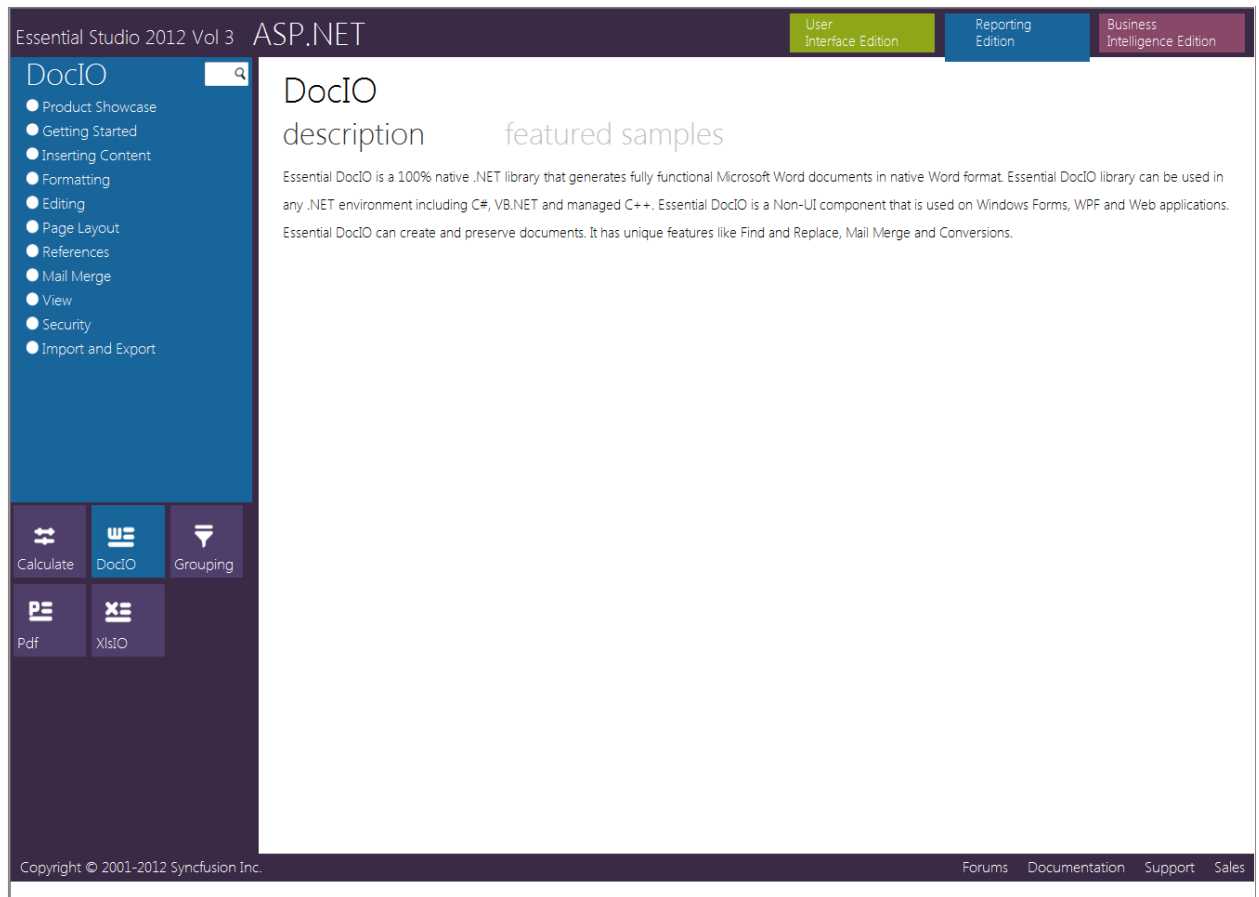


Figure 6: ASP.NET Sample Browser

4. Click **Grouping** from the bottom-left pane. The Grouping samples are displayed.

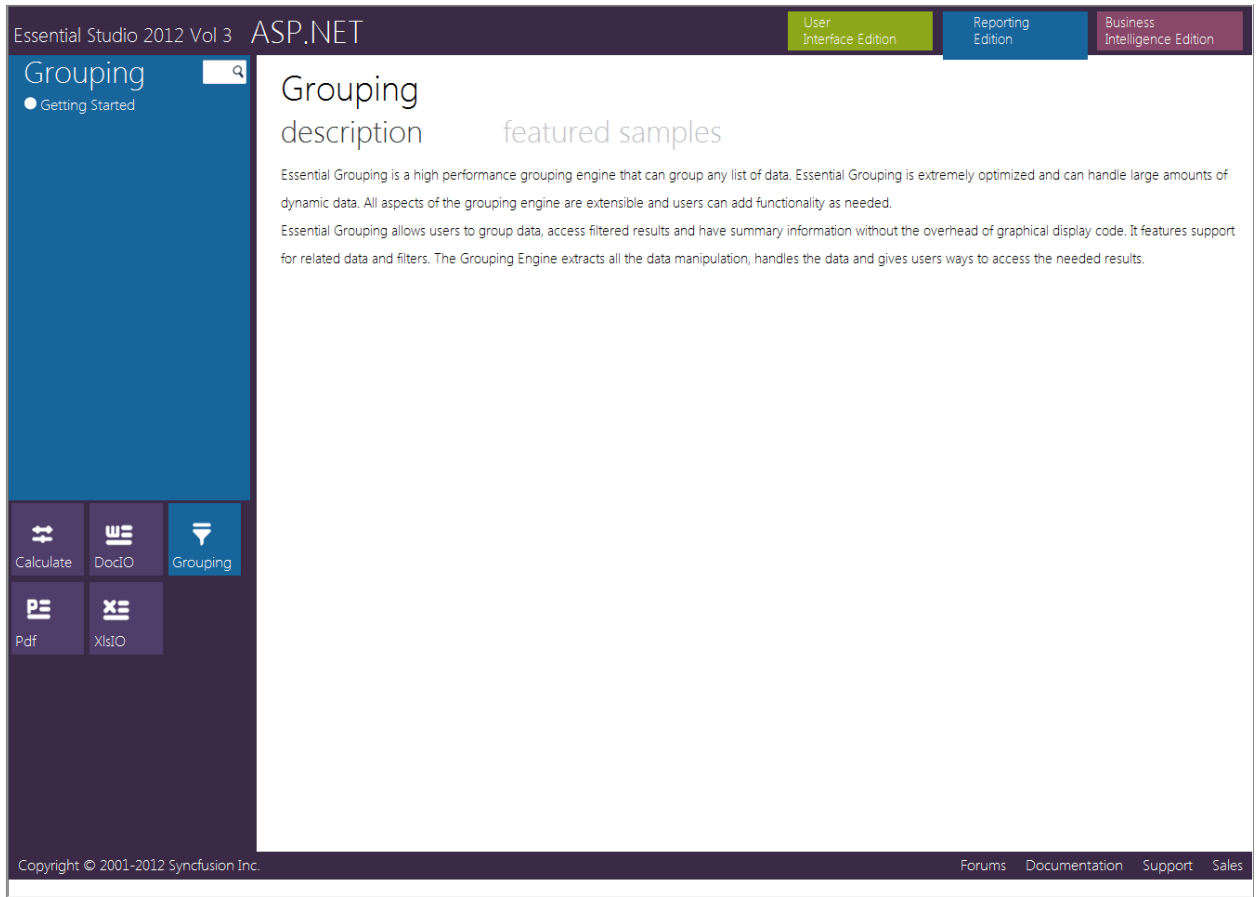


Figure 7: Grouping samples displayed in the ASP.NET Sample Browser

5. Select any sample and browse through the features.

### Source Code Location

#### Windows Forms Source Code

The default location of the Windows Forms Grouping source code is:

**[System Drive]:\Program Files\Syncfusion\Essential Studio\[Version Number]\Windows\Grouping.Windows\Src**

#### ASP.NET Source Code

The default location of the ASP.NET Grouping source code is:

**[System Drive]:\Program Files\Syncfusion\Essential Studio\[Version Number]\Web\Grouping.Web\Src**

## 2.3 Deployment Requirements

### Dll List

While deploying an application that references Syncfusion Essential Grouping assembly, the following dependencies must be included in the distribution.

### Windows Forms – Grouping

- Syncfusion.Core.dll
- Syncfusion.Shared.Base.dll
- Syncfusion.Shared.Windows.dll
- Syncfusion.Grouping.Base.dll
- Syncfusion.Grouping.Windows.dll

### ASP.NET - Grouping

- Syncfusion.Core.dll
- Syncfusion.Shared.Base.dll
- Syncfusion.Shared.Web.dll
- Syncfusion.Grouping.Base.dll
- Syncfusion.Grouping.Windows.dll
- Syncfusion.Grouping.Web.dll

## 3 Getting Started

---

This section will show you how easy it is to get started using Essential Grouping. It will give you a basic introduction to the concepts you need to know before getting started with the product and some tips and ideas on how to implement Grouping into your projects to improve customization and increase efficiency. It shows how to create an `ILog` data source and use it with Grouping. The **datasource** is an **ArrayList** of custom objects. As part of this lesson, you will see how to iterate through the data in the **GroupingEngine**.

### 3.1 Creating Platform Application

This section illustrates the step-by-step procedure to create the following platform applications.

#### Creating a Windows Application

- 1 Open Microsoft Visual Studio. Go to **File** menu and click **New Project**. In the **New Project** dialog, select **Windows Forms Application** template, name the project and click **OK**.

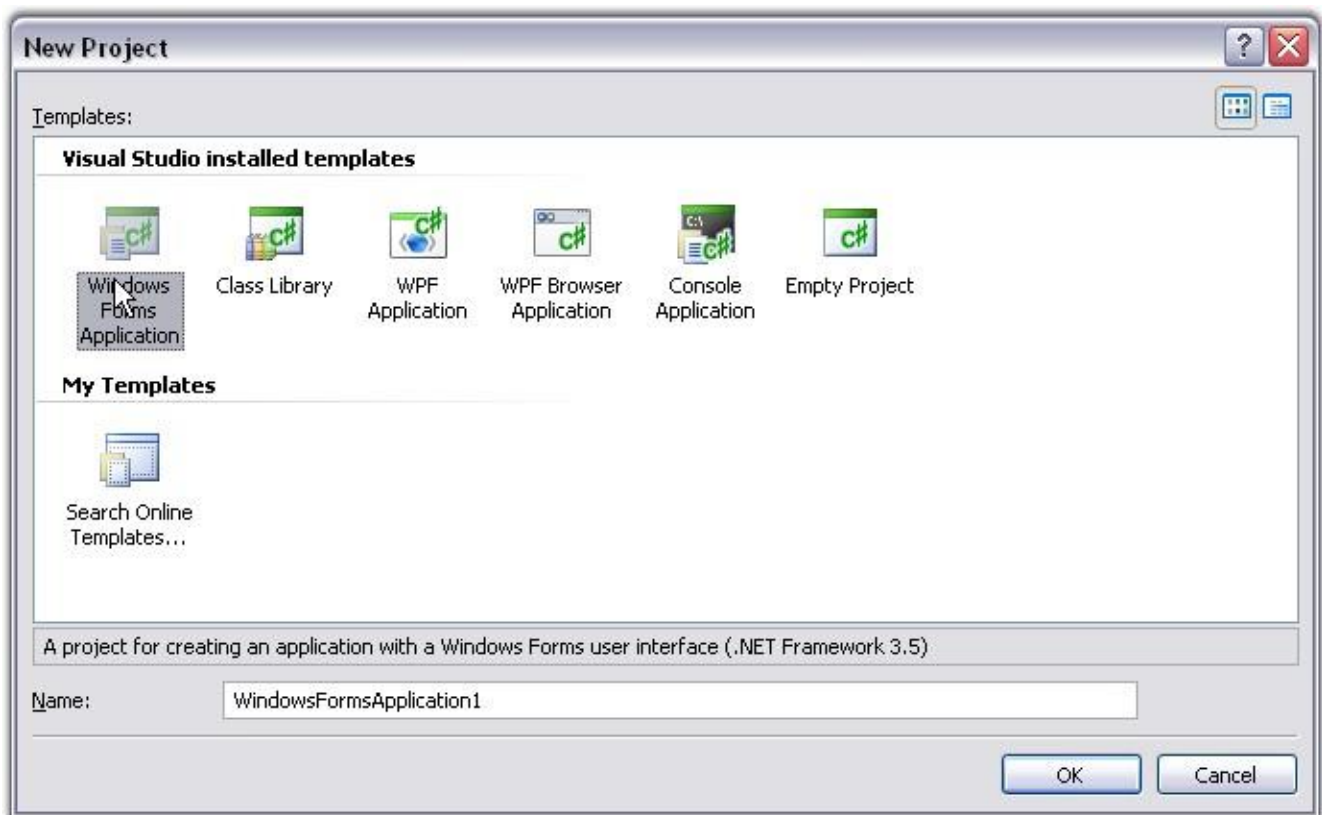




Figure 8: New Project Window

A windows application is created.

6. Now you need to deploy Essential Grouping into this Windows application. Refer Windows / WPF topic for detailed info.

### Creating an ASP.NET Application

To know how to deploy a web application, refer the *ASP.NET Behind the scenes* section in the Getting Started guide of our Essential Studio documentation.

- 1 Open Microsoft Visual Studio. Go to **File** menu and click **New Website**. In the **New Website** dialog, select **ASP.NET Web Site** template, name the website and click **OK**.

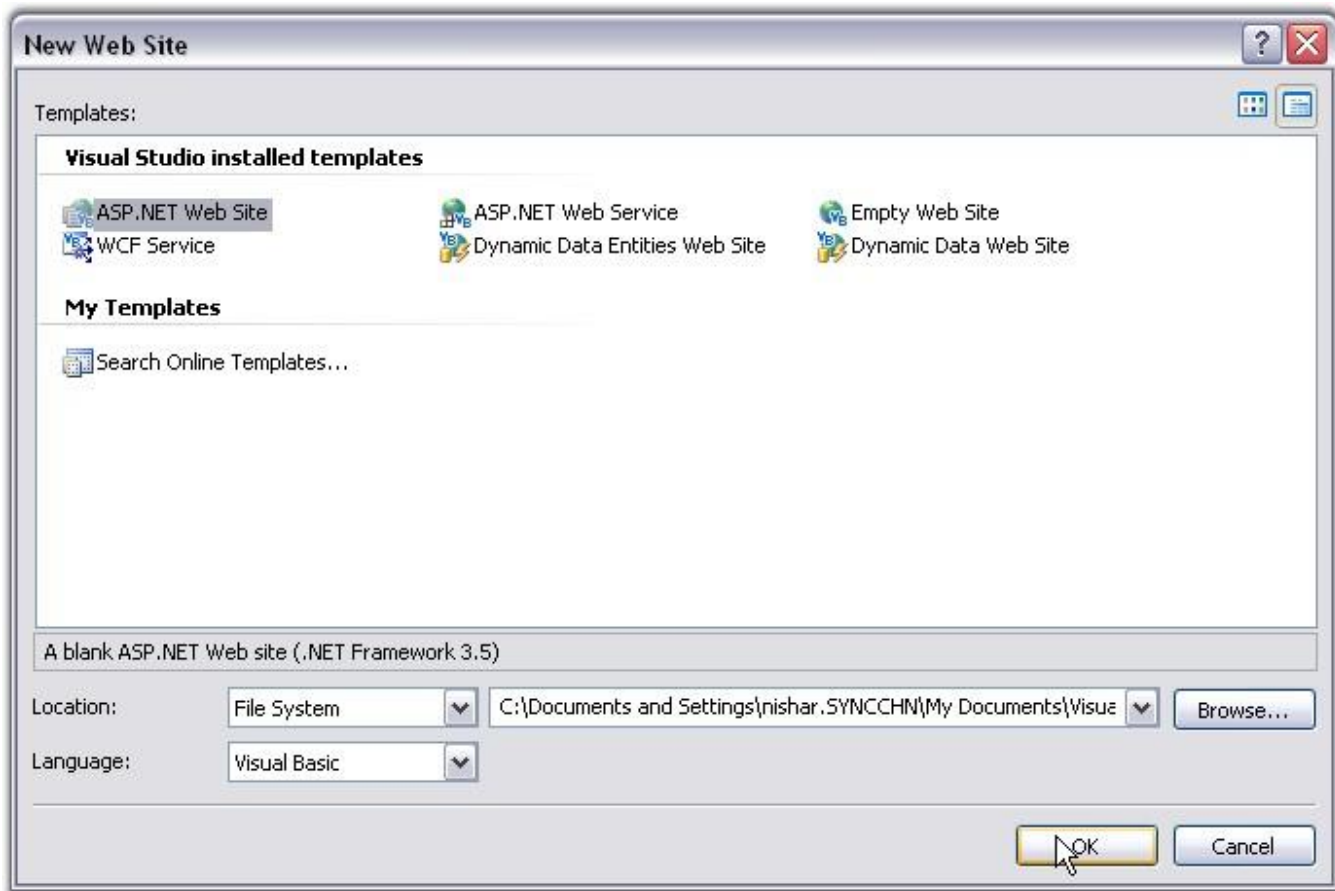


Figure 9: New Web Site dialog box

A web application is created.

7. Now you need to deploy Essential Grouping into this ASP.NET application. Refer ASP.NET topic for detailed info.

### Creating a WPF Application

- 1 Open Microsoft Visual Studio. Go to **File** menu and click **New Project**. In the **New Project** dialog, select **WPF Application** template, name the project and click **OK**.

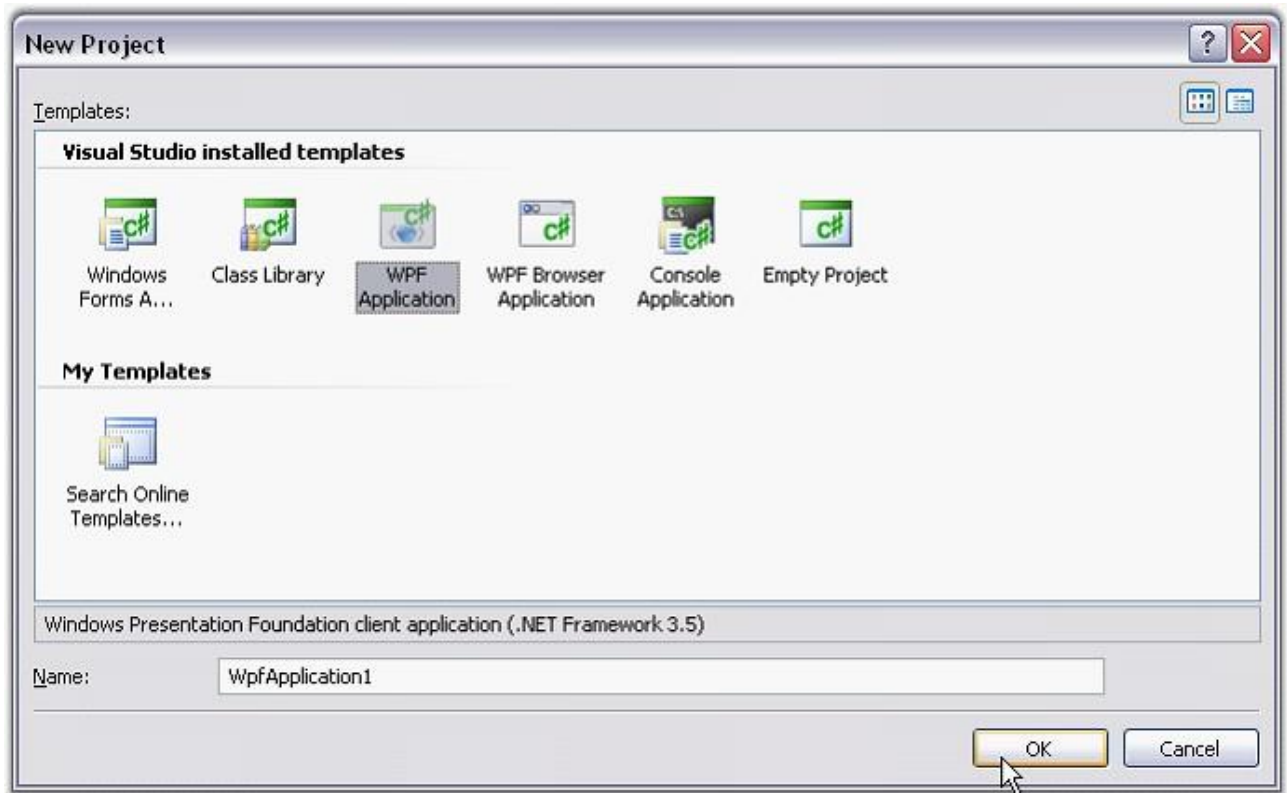


Figure 10: WPF Application

8. A WPF application is created.
9. Now you need to deploy Essential Grouping into this WPF application. Refer Windows / WPF topic for detailed info.

### For More Information Refer:

[Deploying Essential Grouping](#)

## 3.2 Deploying Essential Grouping

We have now created a platform application in the previous topic ([Creating Platform Application](#)). This section will guide you to deploy Essential Grouping in those applications under the following topics:

- Windows / WPF-Step-by-step procedure to deploy Grouping in Windows / WPF applications
- ASP.NET-Step-by-step procedure to deploy Grouping in web application

### 3.2.1 Windows / WPF

Now, you have created a Windows / WPF application (refer [Creating Platform Application](#)). This section will guide you to deploy Essential Grouping in a Windows/WPF applications.

#### Deploying Essential Grouping in a Windows / WPF Application

The following steps will guide you to deploy Essential Grouping:

1. In order to deploy an application that uses the Syncfusion assemblies, the referenced Syncfusion assemblies should reside in the application folder where the exe exists, in the target machine.
2. In order to do that, go to the **References** folder in the **Solution Explorer**. Select all the Syncfusion assemblies, right-click and go to **Properties**. Change the **Copy Local** property of the Syncfusion assemblies to **true** and compile the project.
3. Check whether the licenses.licx file listed in the project has its **Build Action** property to be **Embedded** **Resource**.
4. Now you may see that the Syncfusion assemblies referenced in the project are copied to the output directory along with the application executable (**bin/debug**).
5. Deploy the exe along with the Syncfusion assemblies in that location to the target machine. Be sure that these Syncfusion assemblies reside in the same location as the application exe in the target machine.



**Note:** For Windows Forms applications, placing these referenced Syncfusion assemblies in the GAC alone, in the target machine, will also work.

#### Dlls needed for deployment

- Syncfusion.Core.dll
- Syncfusion.Grouping.Base.dll
- Syncfusion.Grouping.Windows.dll

- Syncfusion.Shared.Base.dll
- Syncfusion.Shared.Windows.dll

Essential Grouping is now deployed in your Windows / WPF applications.

### 3.2.2 ASP.NET

Now, you have created a ASP.NET application (refer [Creating Platform Application](#)). This section will guide you to deploy Essential Grouping in an ASP.NET Application.

The following steps will guide you to deploy Essential Grouping in an ASP.NET application:

1. **Marking the Application directory**-The appropriate directory, usually where the aspx files are stored, must be marked as Application in IIS.
2. Syncfusion **Assemblies**-The Syncfusion assemblies need to be in the **bin** folder that is beside the aspx files.



**Note:** They can also be in the GAC, in which case, they should be referenced in Web.config file.

[Web.config]

```
<configuration>
  <system.web>
    <compilation>
      <assemblies>
<add assembly="Syncfusion.Grid.Grouping.Web, Version=x.x.x.x,
Culture=neutral, PublicKeyToken=3D67ED1F87D44C89"/></assemblies>
      </compilation>
    ...
  </system.web>
</configuration>
```



**Note:** The version numbers in the above references will vary depending on the version you are linking to.

1. **Data Files**-If you have .xml, .mdb, or other data files, ensure that they have sufficient security permission. Authenticated Users should have full control over the files and the directories in order to give ASP.NET code, enough permissions to open the file during runtime.

Refer to the document in the following path, for step by step process of Syncfusion assemblies' deployment in ASP.NET.

[http://www.syncfusion.com/support/user/uploads/webdeployment\\_c883f681.pdf](http://www.syncfusion.com/support/user/uploads/webdeployment_c883f681.pdf)



**Note:** *Application with Essential Grouping needs the following dependent assemblies for deployment.*

- Syncfusion.Shared.Base.dll
- Syncfusion.Shared.Web.dll
- Syncfusion.Grid.Base.dll
- Syncfusion.Grid.Windows.dll
- Syncfusion.Grouping.Base.dll
- Syncfusion.Grouping.Web.dll
- Syncfusion.Grid.Grouping.Base.dll

Essential Grouping is now deployed in your ASP.NET application.

## 4 Concepts and Features

---

This section comprises the following subsections:

- [Data Binding](#)-This section elaborates on the procedure to setup a datasource for the Grouping engine.
- [Using Grouping](#)-In this section, you will see how to group the data, add summaries and locate a particular summary value for a particular group.
- [Data Manipulation](#)-This section will give you information about the additional concepts that are necessary for the complete customization of a spreadsheet. You will also learn about data validation, macros / VBA support, and named ranges. Also included is important information on protection, how to read contents, and implementing ASP.NET usage.
- [Algebra Supported in Expressions / Filters](#)

### 4.1 Data Binding

Essential Grouping lets you sort, group and summarize data. The data needs to be an IList object. For this lesson, we will use an ArrayList of custom objects which have four public properties: A, B, C, and D.

The below section illustrates how to access the data that is bound to the grouping engine.

- [Iterating Through the Data](#)

This section elaborates on the procedure to setup a datasource for the Grouping engine in the below topics.

#### 4.1.1 Creating an ArrayList of Objects

The first thing you need to do is to derive a class that will serve as your custom object.

1. In Visual Studio .NET, select Files -> New -> Project. Then using either C# or VB.NET, select the Console Application project template to create a new Console Application, and name it GroupingSample.

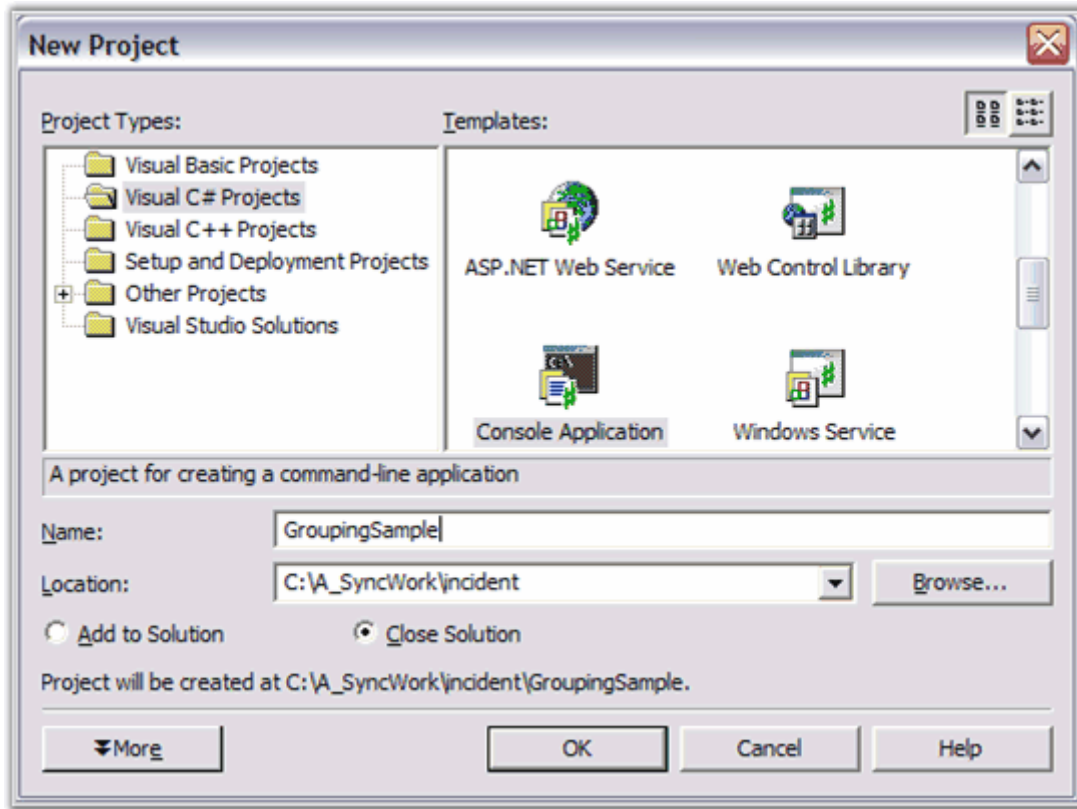


Figure 11: Creating a New Console Project

2. Create a custom object class and add it to the 'Class1' file generated by Step 1. Name the class 'MyObject', and let it have four string properties named A, B, C and D. Also add a constructor that accepts an integer argument. Given below is the sample code for this. Note that you are making B a property only to hold digits. You will be using this property later to illustrate summarizing of data.

```
[C#]

using System;

namespace GroupingSample
{
    class Class1
    {
        [STAThread]
        static void Main(string[] args)
        {
        }
    }

    public class MyObject
```

```
{  
    private string aValue;  
    private string bValue;  
    private string cValue;  
    private string dValue;  
  
    public MyObject(int i)  
    {  
        aValue = string.Format("a{0}", i);  
  
        // Use digit only.  
        bValue = string.Format("{0}", i);  
        cValue = string.Format("c{0}", i%3);  
        dValue = string.Format("d{0}", i%2);  
    }  
    public string A  
    {  
        get{return aValue;}  
        set{aValue = value;}  
    }  
    public string B  
    {  
        get{return bValue;}  
        set{bValue = value;}  
    }  
    public string C  
    {  
        get{return cValue;}  
        set{cValue = value;}  
    }  
    public string D  
    {  
        get{return dValue;}  
        set{dValue = value;}  
    }  
    public override string ToString()  
    {  
        return A + "\t" + B + "\t" + C + "\t" + D;  
    }  
}
```

**[VB.NET]**

Module Module1



```
Sub Main()

End Sub

Public Class MyObject
    Private aValue As String
    Private bValue As String
    Private cValue As String
    Private dValue As String

    Public Sub New(ByVal i As Integer)
        aValue = String.Format("a{0}", i)

        ' Use digit only.
        bValue = String.Format("{0}", i)
        cValue = String.Format("c{0}", i Mod 3)
        dValue = String.Format("d{0}", i Mod 2)
    End Sub 'New

    Public Property A() As String
        Get
            Return aValue
        End Get
        Set(ByVal Value As String)
            aValue = Value
        End Set
    End Property
    Public Property B() As String
        Get
            Return bValue
        End Get
        Set(ByVal Value As String)
            bValue = Value
        End Set
    End Property
    Public Property C() As String
        Get
            Return cValue
        End Get
        Set(ByVal Value As String)
            cValue = Value
        End Set
    End Property
    Public Property D() As String
        Get
            Return dValue
        End Get
    End Property
End Class
```

```
        End Get
        Set(ByVal Value As String)
            dValue = Value
        End Set
    End Property
    Public Overrides Function ToString() As String
        Return A + ControlChars.Tab + B + ControlChars.Tab + C +
ControlChars.Tab + D

        ' ToString
    End Function

    ' MyObject
End Class

End Module
```

3. Add the code to the **Main** function as follows. This creates a random list of 'MyObject' and echoes this list to the console.

### [C#]

```
[STAThread]
static void Main(string[] args)
{
    // Create an arraylist of random MyObjects.
    ArrayList list = new ArrayList();
    Random r = new Random();

    for(int i = 0; i < 10; i++)
    {
        list.Add(new MyObject(r.Next(5)));
        Console.WriteLine(list[i]);
    }

    // Pause
    Console.ReadLine();
}
```

### [VB.NET]

```
Sub Main()

    ' Create an arraylist of random MyObjects.
    Dim list As New ArrayList()
    Dim r As New Random()
```

```
Dim i As Integer
For i = 0 To 10
    list.Add(New MyObject(r.Next(5)))
    Console.WriteLine(list(i))
Next

' Pause
Console.ReadLine()
End Sub
```

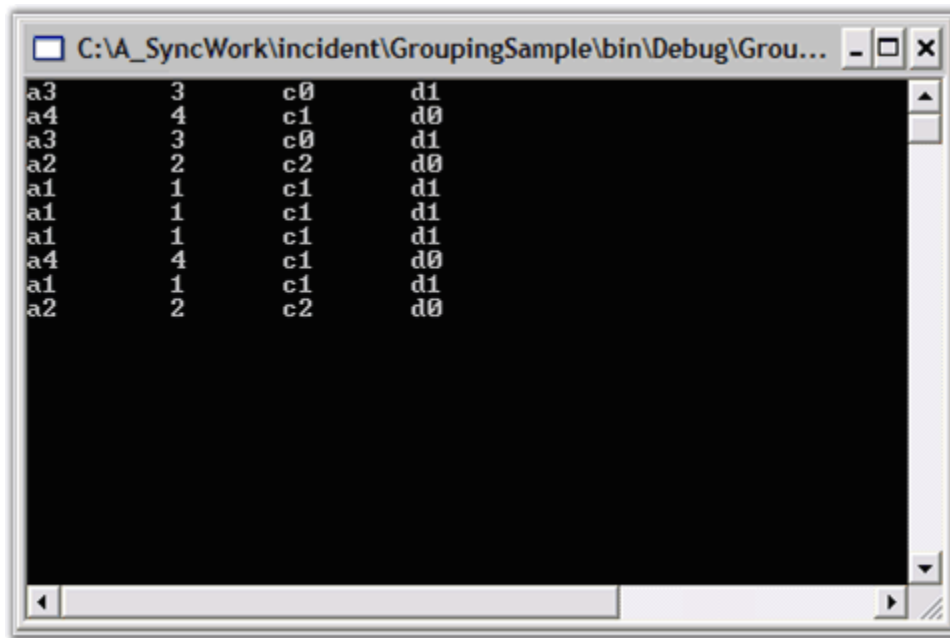



Figure 12: Console Display Showing Random MyObjects

4. An ArrayList of Objects is created.

### 4.1.2 Setting a Datasource In the Grouping Engine

Add the following grouping namespace for referring the assemblies deployed in the application.

 Refer Deploying Essential Grouping section to know about deploying Essential Grouping.


[C#]

```
using Syncfusion.Grouping;
```

[VB.NET]

```
Imports Syncfusion.Grouping
```

Then create a **Grouping.Engine** object and set the ArrayList we created to be its data source.

 For more details on creating array list, refer [Creating an ArrayList Of Objects](#) topic.

[C#]

```
// Put this code in the Main function after Console.ReadLine().  
// Create a Grouping.Engine object.  
Engine groupingEngine = new Engine();  
  
// Set its data source.  
groupingEngine.SetSourceList(list);
```

[VB.NET]

```
Imports Syncfusion.Grouping  
'....  
  
' Put this code in the Main function after Console.ReadLine().  
' Create a Grouping.Engine object.  
Dim groupingEngine As New Engine()  
  
' Set its data source.  
groupingEngine.SetSourceList(list)
```


ArrayList of Objects is set as the datasource for the Grouping engine.

### 4.1.3 Iterating Through the Data

Now, we have set a datasource in the Grouping Engine. Lets see how to iterate through the data.

This section will show you how to access the data through the **Grouping.Engine** object by using the **Engine.Table.Records** collection.

Add the following code to the main function. This code will iterate through the **Records** collection and will display the output in the Console.

 Console is a text-only user interface that allows the user to interact with the operating system or text-based application by entering the text through the keyboard and reading the text output from the computer screen.

### [C#]

```
// Access the data directly from the Engine.
foreach (Record rec in groupingEngine.Table.Records)
{
    MyObject obj = rec.GetData() as MyObject;
    if (obj != null)
    {
        Console.WriteLine(obj);
    }
}

// Pause
Console.ReadLine();
```

### [VB.NET]

```
' Access the data directly from the Engine.
Dim rec As Record
For Each rec In groupingEngine.Table.Records
    Dim obj As MyObject = CType(rec.GetData(), MyObject)
    If Not (obj Is Nothing) Then
        Console.WriteLine(obj)
    End If
Next rec

' Pause
Console.ReadLine()
```

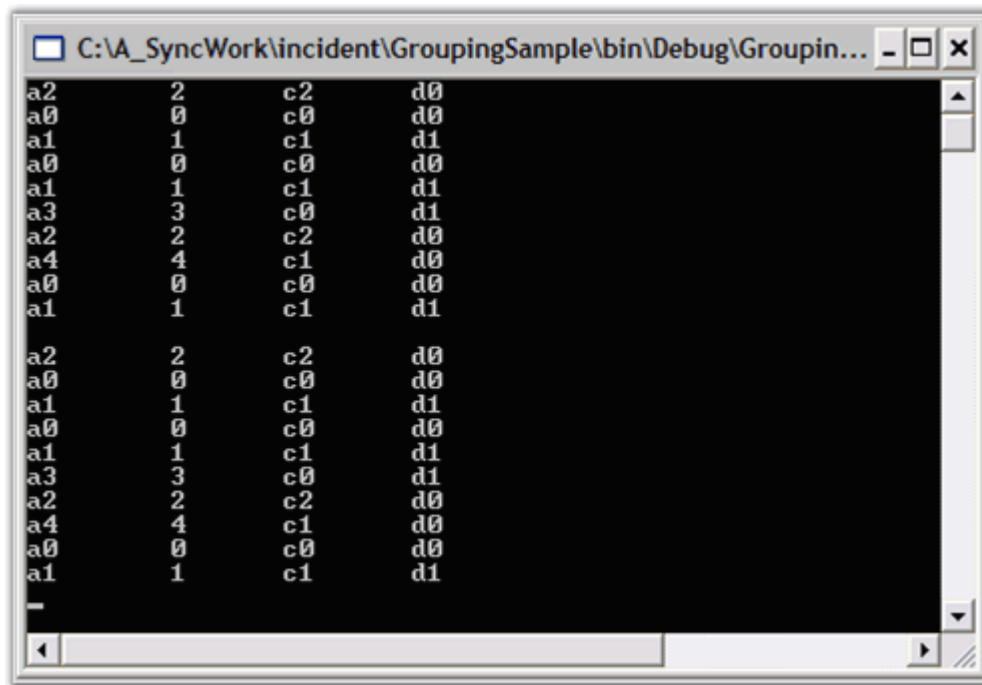


Figure 13: Console Showing Two Sets of Output (Directly from the List and from the Engine)

## 4.2 Using Grouping

The Grouping of data is one type of data analysis technique. It is natural to organize data into groups. For example, you may want to group your sales details by months and get the total sales on a month-by-month basis. The following sections elaborates on this:

### 4.2.1 Grouping a Table

In this lesson, you will start working with the **Grouping.Engine** object to see how to apply a grouping to the data as well as summarize the data. In the [Data Binding](#) section, you used the **grouping.Engine.Table.Records** collection to access the data in the Grouping.Engine object. The **grouping.Engine.Table** property is the property of the Grouping.Engine that holds the actual data needed by Essential Grouping.

You will now look at the property that holds the schema information that is associated with the data, i.e., the **grouping.Engine.TableDescriptor** property. For example, the **TableDescriptor.Columns** property holds a collection of ColumnDescriptor objects that define the schema information on the columns in the data.

**Note:** Here, the columns correspond to the public properties in our sample *MyObject* class, *A*, *B*, *C*, and *D*.

We will now continue using the same sample created in the [previous](#) section and add the corresponding code at the bottom of the *Main* method.

1. To group the 'MyObject' ArrayList by a particular property, say property C, you have to add only the property name ("C") to the **grouping.Engine.TableDescriptor.GroupedColumns** collections. Add the following code snippet to the bottom of the **Main** method.

[C#]

```
// Group on property C.
groupingEngine.TableDescriptor.GroupedColumns.Add("C");

// Display the records in the engine after grouping.
foreach (Record rec in groupingEngine.Table.Records)
{
    MyObject obj = rec.GetData() as MyObject;
    if (obj != null)
    {
        Console.WriteLine(obj);
    }
}
```

[VB.NET]

```
' Group on property C.
groupingEngine.TableDescriptor.GroupedColumns.Add("C")

' Display the records in the engine after grouping.
For Each rec In groupingEngine.Table.Records
    Dim obj As MyObject = CType(rec.GetData(), MyObject)
    If Not (obj Is Nothing) Then
        Console.WriteLine(obj)
    End If
Next rec
```

2. After running the code from step 1, a screen similar to the one below will be displayed. Note that the bottom list displayed is now sorted by **column C**. This is a one side effect of grouping by column C.

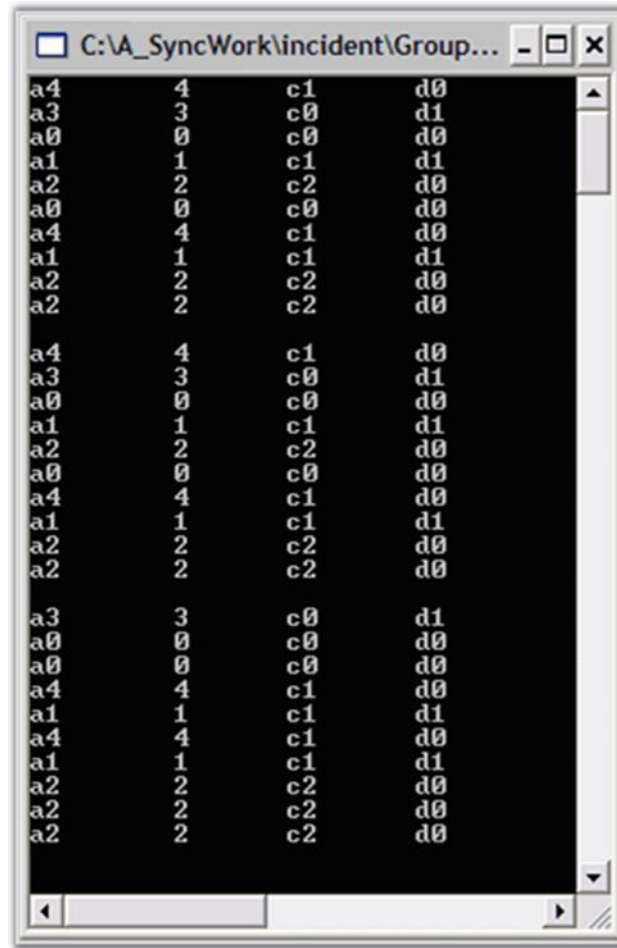


Figure 14: Display After Grouping by Property C

#### 4.2.1.1 The Grouping.TableDescriptor Class

As noted previously, the **grouping.TableDescriptor** is the property that maintains the schema information for the data source. Here is a table showing some collections in the **TableDescriptor** that you will be using as the lessons continue.

Table 5: TableDescriptor Property

TableDescriptor Property	Description
SortedColumns	Holds sorted properties.
GroupedColumns	Holds grouped properties.
Summaries	Holds the columns that have summaries.
RecordFilters	Holds information on columns that are part of



	filters.
SortedColumns	Holds sorted properties.

## 4.2.2 Accessing a Particular Group

Grouping is a recursive process whereby a data source may be grouped several times. This leads to the recursive situation of groups having sub-groups and so on. In recursion, there is usually some primary node or initial starting point that you use, to work with the recursive objects. In Grouping, the initial starting point is **Engine.Table.TopLevelGroup**. This is the 'primary' Group object.

The **Grouping.Group** class has two properties that are used to recursively access nested groups and the Record objects contained in the terminal group. The properties are:

- Group.Groups and Group.Records.
1. Group.Groups is a collection of Group objects that are contained in the parent Group, and Group.Records is a collection of Records that are contained in the parent Group.
  2. At most one of these collections will actually hold objects. If the **Groups** collections is populated, this implies that the Group has sub-groups and there are no records.
  3. If the **Records** collection is populated then, it implies that this Group is a terminal group with records, but there are no sub-groups.

Your first task is to add a recursive method to either display records if the Group has records, or to recursively call itself to display any records of its child groups.

1. Add the following code below the **Main** method to implement a recursive method to display records in a Group.

```
[C#]

private static void ShowRecordsUnderGroup(Group g)
{
    if(g.Records != null && g.Records.Count > 0)
    {

        // Displaying the data for all the records in each group.
        foreach(Record rec in g.Records)
        {
            MyObject obj = rec.GetData() as MyObject;
            if(obj != null)
```

```
        {
            Console.WriteLine(obj);
        }
        Console.WriteLine("--");
    }
    else if(g.Groups != null && g.Groups.Count > 0)
    {

        // Iterating through the groups.
        foreach(Group g1 in g.Groups)
        {

            // Recursive call
            ShowRecordsUnderGroup(g1);
        }
    }
}
```

### [VB.NET]

```
Private Sub ShowRecordsUnderGroup(ByVal g As Group)
    If Not (g.Records Is Nothing) And g.Records.Count > 0 Then
        Dim rec As Record

        ' Displaying the data for all the records in each group.
        For Each rec In g.Records
            Dim obj As MyObject = CType(rec.GetData(), MyObject)
            If Not (obj Is Nothing) Then
                Console.WriteLine(obj)
            End If
        Next rec
        Console.WriteLine("--")
    Else
        If Not (g.Groups Is Nothing) And g.Groups.Count > 0 Then
            Dim g1 As Group

            ' Iterating through the groups.
            For Each g1 In g.Groups

                ' Recursive call
                ShowRecordsUnderGroup(g1)
            Next g1
        End If
    End If

    ' ShowRecordsUnderGroup
End Sub
```

2. Once you have your **ShowRecordsUnderGroup** method, you only have to retrieve a particular group from the Groups collection and then call the method. So, after grouping on property C, you can view all the records whose Category is "c1" using the code like the one given below the Main method.

**[C#]**

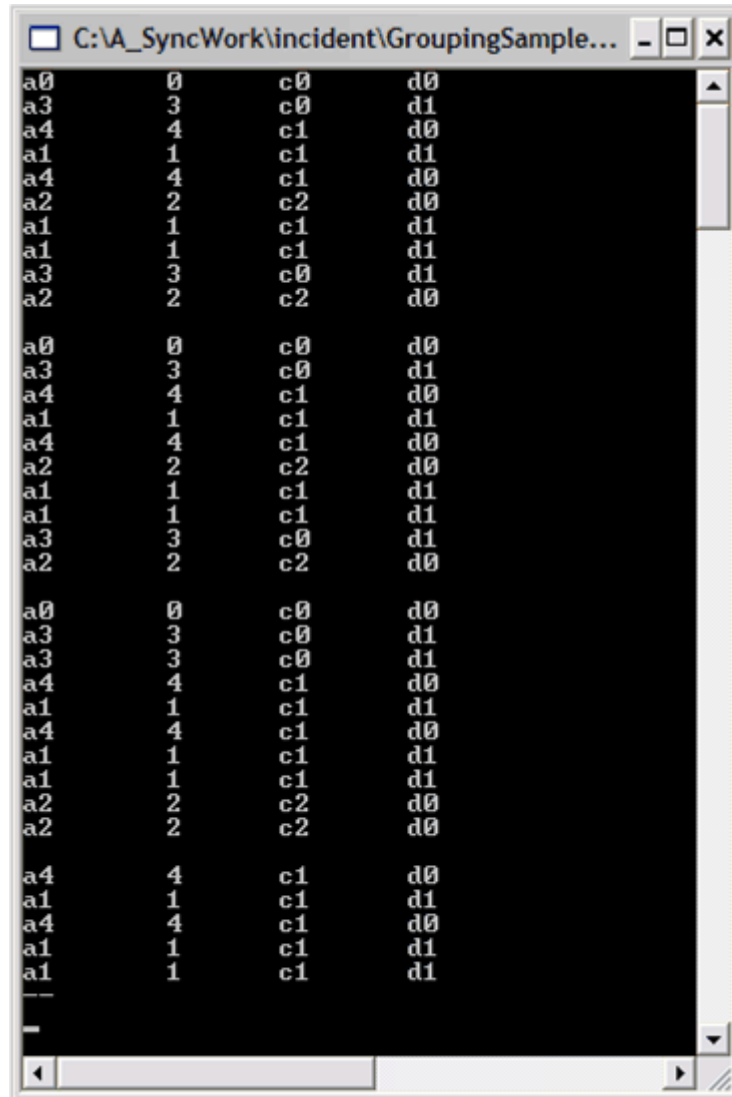
```
// Get the Group associated with the value "c1".
Group g = groupingEngine.Table.TopLevelGroup.Groups["c1"];
ShowRecordsUnderGroup(g);

// Pause
Console.ReadLine();
```

**[VB.NET]**

```
' Get the Group associated with the value "c1".
Dim g As Group = groupingEngine.Table.TopLevelGroup.Groups("c1")
ShowRecordsUnderGroup(g)

' Pause
Console.ReadLine()
```



a0	0	c0	d0
a3	3	c0	d1
a4	4	c1	d0
a1	1	c1	d1
a4	4	c1	d0
a2	2	c2	d0
a1	1	c1	d1
a1	1	c1	d1
a3	3	c0	d1
a2	2	c2	d0
a0	0	c0	d0
a3	3	c0	d1
a4	4	c1	d0
a1	1	c1	d1
a4	4	c1	d0
a2	2	c2	d0
a1	1	c1	d1
a1	1	c1	d1
a3	3	c0	d1
a2	2	c2	d0
a0	0	c0	d0
a3	3	c0	d1
a3	3	c0	d1
a4	4	c1	d0
a1	1	c1	d1
a4	4	c1	d0
a1	1	c1	d1
a1	1	c1	d1
a2	2	c2	d0
a2	2	c2	d0
a4	4	c1	d0
a1	1	c1	d1
a4	4	c1	d0
a1	1	c1	d1
a1	1	c1	d1
--			
-			

Figure 15: Last Section Shows the Records from the "c1" Group

3. Similar code can be used to display all the records by passing the 'primary' group to your ShowRecordsUnderGroup method. To implement this, add the following code to the Main method.

```
[C#]

// Show all records under the TopLevelGroup.
ShowRecordsUnderGroup(groupingEngine.Table.TopLevelGroup);

// Pause
Console.ReadLine();
```

[VB.NET]

```
' Show all records under the TopLevelGroup.
ShowRecordsUnderGroup (groupingEngine.Table.TopLevelGroup)

' Pause
Console.ReadLine()
```

### 4.2.3 Adding a Summary

Essential Grouping lets you summarize your data by adding **SummaryDescriptor** objects to the schema information that is stored in the **Engine.TableDescriptor.Summaries** collection. You can have multiple summaries by adding several **SummaryDescriptors**.

At the bottom of the Main method, add this code to create a summary item for the Engine.

[C#]


```
// Create a summary that computes the Int32Aggregate calculations on property
B.
SummaryDescriptor sdBInt32Agg = new SummaryDescriptor("BInt32Agg", "B",
SummaryType.Int32Aggregate);

// Add this summary to the Summaries collection.
groupingEngine.TableDescriptor.Summaries.Add(sdBInt32Agg);
```

[VB.NET]

```
' Create a summary that computes the Int32Aggregate calculations on property
B.
Dim sdBInt32Agg As New SummaryDescriptor("BInt32Agg", "B",
SummaryType.Int32Aggregate)

' Add this summary to the Summaries collection.
groupingEngine.TableDescriptor.Summaries.Add(sdBInt32Agg)
```

 There are several overloads of the constructor for **SummaryDescriptor**. Here, we are using the overload that accepts a **SummaryType** enum as the third argument. This **SummaryType** will allow you to pick out some predefined calculations such as the **Int32Aggregate** functions like **Max**, **Min**, **Sum**, and **Average**. There are enums that specify double, boolean, and other aggregate types. Here, we choose **Int32** as that is the type of value you will see in the B property in the data.

## 4.2.4 Retrieving Summary Values for a Particular Group

After adding the **SummaryDescriptor**, the **GroupingEngine** will maintain these summary items in a group by group basis. You can access these summary values for any group. Here we will get the values for the **TopLevelGroup** and for our **c1** group to illustrate how this is done.

To obtain a particular group's summary values, you need to get the group and access the summary through its **GetSummary** method. Once you have the summary, you can cast it to its **Int32AggregateSummary** type.

The following code snippet illustrates this.

**[C#]**

```
// To simplify notation, add this statement at the top of the file.
using ISummary = Syncfusion.Collections.BinaryTree.ITreeTableSummary;

// At the bottom of the Main method, add these lines.

// Now go through the group to get the Summary value for the group.
ISummary groupSummary =
groupingEngine.Table.TopLevelGroup.GetSummary("BInt32Agg");
Int32AggregateSummary int32Summary = (Int32AggregateSummary) groupSummary;

Console.WriteLine("whole table {0}, {1}, {2}", int32Summary.Sum,
int32Summary.Average, int32Summary.Maximum);

// Value for "c1" group.
groupSummary =
groupingEngine.Table.TopLevelGroup.Groups["c1"].GetSummary("BInt32Agg");
int32Summary = (Int32AggregateSummary) groupSummary;

Console.WriteLine("c1-group {0}, {1}, {2}", int32Summary.Sum,
int32Summary.Average, int32Summary.Maximum);

// Pause
Console.ReadLine();
```

**[VB.NET]**

```
' At the bottom of the Main method, add these lines.

' Now go through the group to get the Summary value for the group.
```

```
Dim groupSummary As Syncfusion.Collections.BinaryTree.ITreeTable =
groupingEngine.Table.TopLevelGroup.GetSummary("BInt32Agg")

Dim int32Summary As Int32AggregateSummary = CType(groupSummary,
Int32AggregateSummary)
Console.WriteLine("whole table {0}, {1}, {2}", int32Summary.Sum,
int32Summary.Average, int32Summary.Maximum)

' Value for "c1" group.
groupSummary =
groupingEngine.Table.TopLevelGroup.Groups("c1").GetSummary("BInt32Agg")
int32Summary = CType(groupSummary, Int32AggregateSummary)

Console.WriteLine("c1-group {0}, {1}, {2}", int32Summary.Sum,
int32Summary.Average, int32Summary.Maximum)

' Pause
Console.ReadLine()
```

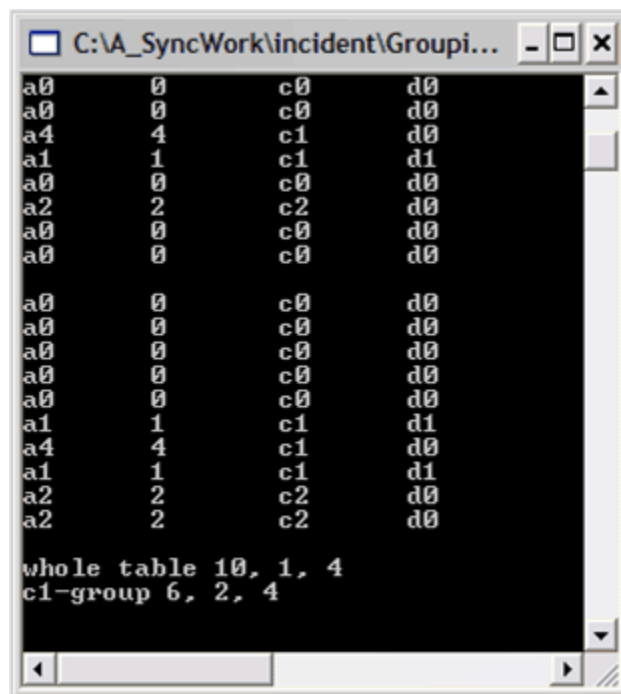


Figure 16: Summary Statistics Shown for the Whole Table and for Group c1

## 4.3 Data Manipulation

In addition to grouping data, you may want to filter it for some special criteria. For example, you may want to see the total monthly sales due to orders under some value. Essential Grouping gives you the flexibility to add calculated values to the data, and then use these values to produce other information like total monthly sales due for respective order etc.

The following data manipulation techniques are available in Essential Grouping:

### 4.3.1 Filters

Another collection that is part of the schema information in the **Engine.TableDescriptor** is the **RecordFilters** collection. This collection lets you filter the data to see only the items that are in your data list and that satisfy the criteria that is specified. You can express the criteria as a logical expression using the property names, algebraic, and logical operators. You can also use LIKE, MATCH, and IN operators.

- 1 In the Console Application used in lessons 1 and 2, comment out all the code that is in the **Main** method and add the following code to create a data object and set it into the **Grouping Engine**.

[C#]

```
static void Main(string[] args)
{
    // Create an arraylist of random MyObjects.
    ArrayList list = new ArrayList();
    Random r = new Random();

    for(int i = 0; i < 10; i++)
    {
        list.Add(new MyObject(r.Next(5)));
    }

    // Create a Grouping.Engine object.
    Engine groupingEngine = new Engine();

    // Set its datasource.
    groupingEngine.SetSourceList(list);
}
```

[VB.NET]



```
Sub Main()  
  
    ' Create an arraylist of random MyObjects.  
    Dim list As New ArrayList()  
    Dim r As New Random()  
  
    Dim i As Integer  
    For i = 0 To 10  
        list.Add(New MyObject(r.Next(5)))  
    Next  
  
    ' Create a Grouping.Engine object.  
    Dim groupingEngine As New Engine()  
  
    ' Set its data source.  
    groupingEngine.SetSourceList(list)  
End Sub
```

10. You are now ready to apply a filter to the data. Say for example you want to see only those items whose property D has the value d1. You must observe that D is a string that has non-numeric values. So, in this case you will need to use one of the string comparison operators (LIKE or MATCH) in your filter condition.
11. To add a filter condition, you will need to add a **RecordFilterDescriptor** to the **Engine.TableDescriptor.RecordFilters** collection.

### Do the following steps:

1. List the data before the filter.
2. Apply the filter by creating a RecordFilterDescriptor.
3. Add it to the RecordFilters collection.
4. List the filtered data.



**Note:** To list the data, instead of accessing the *Table.Records* collections, you were using the *Table.FilteredRecords* collections. The *FilteredRecords* collection only includes the records that satisfy all filters in the *RecordFilters* collection. Add this code at the end of the *Main* method.

12. Note that the constructor on the *RecordFilterDescription* takes an expression, "[D] LIKE 'd1'". This expression will be **True** only for those items in the list where property D has the value d1.

[C#]

```
// Display the data before filtering.
```

```
foreach (Record rec in groupingEngine.Table.FilteredRecords)
{
    MyObject obj = rec.GetData() as MyObject;
    if (obj != null)
    {
        Console.WriteLine(obj);
    }
}

// Pause
Console.ReadLine();

// Filter on [D] = d1
RecordFilterDescriptor rfd = new RecordFilterDescriptor("[D] LIKE 'd1'");
groupingEngine.TableDescriptor.RecordFilters.Add(rfd);

// Display the data after filtering.
foreach (Record rec in groupingEngine.Table.FilteredRecords)
{
    MyObject obj = rec.GetData() as MyObject;
    if (obj != null)
    {
        Console.WriteLine(obj);
    }
}

// Pause
Console.ReadLine();
```

### [VB.NET]

```
' Display the data before filtering.
Dim rec As Record
    For Each rec In groupingEngine.Table.FilteredRecords
Dim obj As MyObject = CType(rec.GetData(), MyObject)
        If Not (obj Is Nothing) Then
            Console.WriteLine(obj)
        End If
    Next rec

' Pause
Console.ReadLine()

' Filter on [D] = d1
Dim rfd As New RecordFilterDescriptor("[D] LIKE 'd1'")
```

```

groupingEngine.TableDescriptor.RecordFilters.Add(rfd)

' Display the data after filtering.
For Each rec In groupingEngine.Table.FilteredRecords
Dim obj As MyObject = CType(rec.GetData(), MyObject)
    If Not (obj Is Nothing) Then
        Console.WriteLine(obj)
    End If
Next rec

' Pause
Console.ReadLine()

```

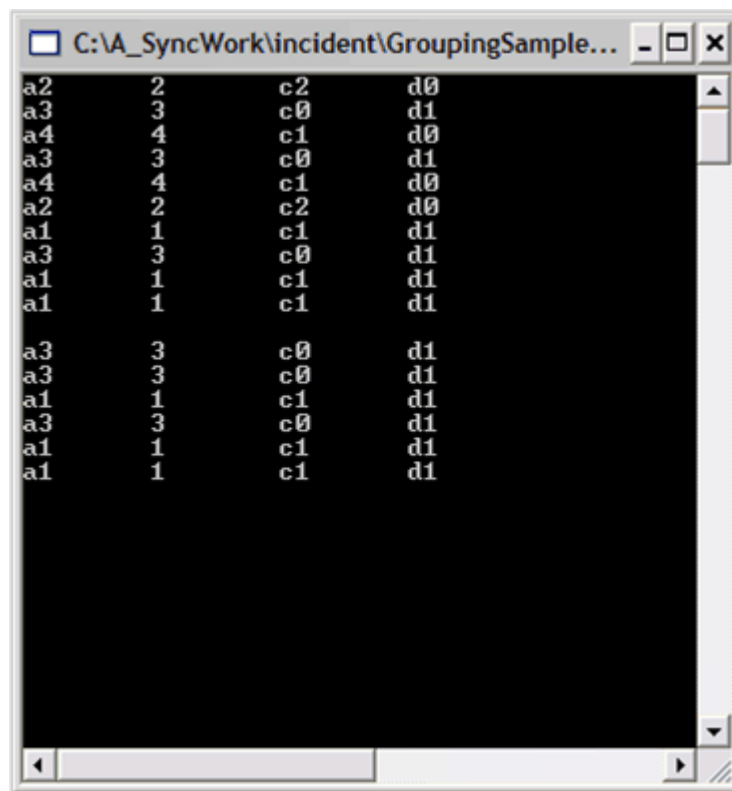


Figure 17: Screen Showing the Initial Data Followed by the Same Data Filtered on [D] LIKE 'd1'

13. You can apply more complex filters. Here is the code that will remove any existing filters and filter the property D being d1 or property b equal 2. Note here that since you expect property B to display only numeric data you must use the = operator in the comparison.

[C#]

```
groupingEngine.TableDescriptor.RecordFilters.Clear();
```

```
rfd = new RecordFilterDescriptor("[D] LIKE 'd1' OR [B] = 2");
groupingEngine.TableDescriptor.RecordFilters.Add(rfd);

// Access the data directly from the Engine.
foreach (Record rec in groupingEngine.Table.FilteredRecords)
{
    MyObject obj = rec.GetData() as MyObject;
    if (obj != null)
    {
        Console.WriteLine(obj);
    }
}

// Pause
Console.ReadLine();
```

### [VB.NET]

```
groupingEngine.TableDescriptor.RecordFilters.Clear()

rfd = New RecordFilterDescriptor("[D] LIKE 'd1' OR [B] = 2")
groupingEngine.TableDescriptor.RecordFilters.Add(rfd)

' Access the data directly from the Engine.
For Each rec In groupingEngine.Table.FilteredRecords
Dim obj As MyObject = CType(rec.GetData(), MyObject)
    If Not (obj Is Nothing) Then
        Console.WriteLine(obj)
    End If
Next rec

' Pause
Console.ReadLine()
```

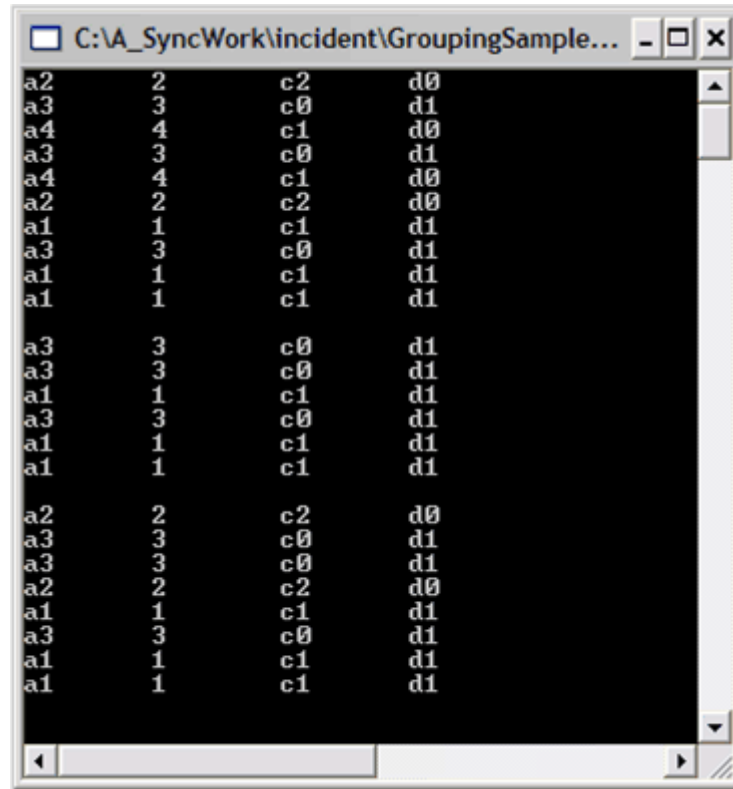


Figure 18: Screen Showing Data Filtered on [D] LIKE 'd1' OR [B] = 2

Filtering is applied to the data displayed in the console.

### 4.3.2 Expressions

You can add new properties to your data object that are algebraic expressions involving other properties of the object.

To add an expression, you need to create an **ExpressionFieldDescriptor** and add it to the **Engine.TableDescriptor.Expression.Fields** collection. Here we illustrate this process by adding an expression that computes 2.1 times the value of property B plus 3.2.

- 1 In the Console Application, comment out all the code that is in the **Main** method and add this code to create a data object and set it into the **GroupingEngine**.

[C#]

```
// Create an arraylist of random MyObjects.
ArrayList list = new ArrayList();
```

```
Random r = new Random();

for(int i = 0; i < 10; i++)
{
    list.Add(new MyObject(r.Next(5)));
}

// Create a Grouping.Engine object.
Engine groupingEngine = new Engine();

// Set its datasource.
groupingEngine.SetSourceList(list);
```

### [VB.NET]

```
' Create an arraylist of random MyObjects.
Dim list As New ArrayList()
Dim r As New Random()

Dim i As Integer
For i = 0 To 10
    list.Add(New MyObject(r.Next(5)))
Next

' Create a Grouping.Engine object.
Dim groupingEngine As New Engine()

' Set its datasource.
groupingEngine.SetSourceList(list)
```

14. Now you must add code to list the data, add an expression property and then display the value of the expression. To retrieve the value, you must use the **Record.GetValue** method by passing it as the name of the expression that you had assigned when it was created.

### [C#]

```
// Display the data before filtering.
foreach(Record rec in groupingEngine.Table.FilteredRecords)
{
    MyObject obj = rec.GetData() as MyObject;
    if(obj != null)
    {
        Console.WriteLine(obj);
    }
}
```

```
// Pause
Console.ReadLine();

    // Add an expression property.
    ExpressionFieldDescriptor efd = new ExpressionFieldDescriptor("MultipleOfB",
        "2.1 * [B] + 3.2");
    groupingEngine.TableDescriptor.ExpressionFields.Add(efd);

// Display the data after adding the field.
foreach (Record rec in groupingEngine.Table.FilteredRecords)
{
    Console.WriteLine(rec.GetValue("MultipleOfB"));
}

// Pause
Console.ReadLine();
```

### [VB.NET]

```
' Display the data before filtering.
Dim rec As Record
For Each rec In groupingEngine.Table.FilteredRecords
    Dim obj As MyObject = CType(rec.GetData(), MyObject)
    If Not (obj Is Nothing) Then
        Console.WriteLine(obj)
    End If
Next rec

' Pause
Console.ReadLine()

' Add an expression property.
Dim efd As New ExpressionFieldDescriptor("MultipleOfB", "2.1 * [B] + 3.2")
groupingEngine.TableDescriptor.ExpressionFields.Add(efd)

' Display the data after adding the field.
For Each rec In groupingEngine.Table.FilteredRecords
    Console.WriteLine(rec.GetValue("MultipleOfB"))
Next rec
```

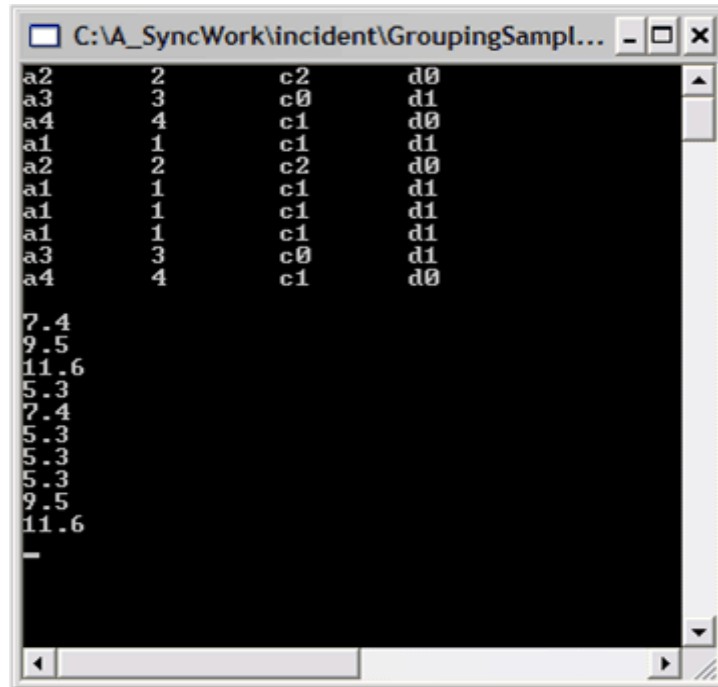


Figure 19: Screen Showing the Initial Data Followed by Values Computed Using the Expression  $2.1 * [B] + 3.2$

### 4.3.3 Sorting

To sort your data, you must add the name of the property that you want to sort to the **Engine.TableDescriptor.SortedColumns** collection.

In the Console Application, comment out all the code that is in the **Main** method and add this code to create a data object, set it into the **GroupingEngine**, display the data, sort it by property A and re-display the sorted data.

```
[C#]

// Create an arraylist of random MyObjects.
ArrayList list = new ArrayList();
Random r = new Random();

for(int i = 0; i < 10; i++)
{
    list.Add(new MyObject(r.Next(5)));
}

// Create a Grouping.Engine object.
Engine groupingEngine = new Engine();
```



```
// Set its datasource.
groupingEngine.SetSourceList(list);

// Display the data before sorting.
foreach (Record rec in groupingEngine.Table.Records)
{
    MyObject obj = rec.GetData() as MyObject;
    if (obj != null)
    {
        Console.WriteLine(obj);
    }
}

// Pause
Console.ReadLine();

// Sort column A.
groupingEngine.TableDescriptor.SortedColumns.Add("A");

// Display the data after sorting.
foreach (Record rec in groupingEngine.Table.FilteredRecords)
{
    MyObject obj = rec.GetData() as MyObject;
    if (obj != null)
    {
        Console.WriteLine(obj);
    }
}

// Pause
Console.ReadLine();
```

### [VB.NET]

```
' Create an arraylist of random MyObjects.
Dim list As New ArrayList()
Dim r As New Random()

Dim i As Integer
For i = 0 To 9
    list.Add(New MyObject(r.Next(5)))
Next i

' Create a Grouping.Engine object.
Dim groupingEngine As New Engine()
```

```
' Set its datasource.
    groupingEngine.SetSourceList(list)

' Display the data before sorting.
Dim rec As Record
For Each rec In groupingEngine.Table.Records
Dim obj As MyObject = CType(rec.GetData(), MyObject)
    If Not (obj Is Nothing) Then
        Console.WriteLine(obj)
    End If
Next rec

' Pause
Console.ReadLine()

' Sort column A.
groupingEngine.TableDescriptor.SortedColumns.Add("A")

' Display the data after sorting.
For Each rec In groupingEngine.Table.FilteredRecords
    Dim obj As MyObject = CType(rec.GetData(), MyObject)
    If Not (obj Is Nothing) Then
        Console.WriteLine(obj)
    End If
Next rec

' Pause
Console.ReadLine()
```

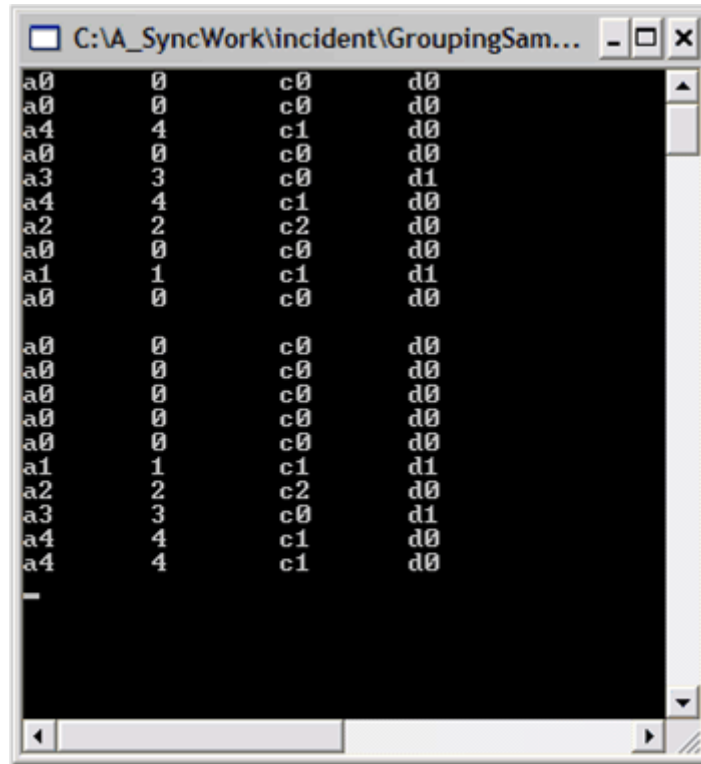


Figure 20: Screen Showing the Data Before and After Sorting on Property A

### 4.3.4 Custom Sorting

We used a simplest overload in the previous section for sorting the data. The **TableDescriptor.SortedColumns.Add** method has three overloads as shown below:

- Add(string propertyName)
- Add(string propertyName, ListSortDirection dir)
- Add(SortColumnDescriptor sdc)

The following code snippets illustrate the syntax for these overloads:

[C#]

```
public int Add(string propertyName);
public int Add(string propertyName, ListSortDirection dir);
public int Add(SortColumnDescriptor sdc);
```

### [VB.NET]

```
Overloads Public Function Add(propertyName As String) As Integer
Overloads Public Function Add(propertyName As String, dir As
ListSortDirection) As Integer
Overloads Public Function Add(sdc As SortColumnDescriptor) As Integer
```

The second overload can be used to specify the sort direction. The **SortColumnDescriptor.Comparer** is an **IComparer** property that allows you to specify a custom comparer object.



**Note:** We are going to use the third function in this section to perform custom sorting.

The following steps illustrate how to do custom sorting using the **IComparer** property:

- 1 Here, you must create a class that implements **IComparer** which can accept value such as *ax*, where *x* is a digit which is used to do the comparison. This leads to numerical sorting, ignoring the leading character.

15. Add this code immediately following the end of the **Class1** code.

### [C#]

```
public class AComparer : IComparer
{
    // Implementing IComparer to define the object comparisons.
    public int Compare(object x, object y)
    {
        if(x == null && y == null)
            return 0;
        else if(x == null)
            return -1;
        else if(y == null)
            return 1;
        else
        {
            int sx = 0;
            int sy = 0;
            try
            {
                // Ignoring the leading character to have numerical
                sorting.
                sx = int.Parse(x.ToString().Substring(1));
            }
            catch { }
        }
    }
}
```

```
        sy = int.Parse(y.ToString().Substring(1));  
        return sy - sx;  
    }  
    catch  
    {  
        throw new ArgumentException("A must be in the form  
'annnn'");  
    }  
}  
}
```

### [VB.NET]

```
Public Class AComparer  
    Implements IComparer  
    ' Implementing IComparer to define the object comparisons.  
    Public Function Compare(ByVal x As Object, ByVal y As Object) As  
Integer _  
        Implements IComparer.Compare  
        If x Is Nothing And y Is Nothing Then  
            Return 0  
        Else  
            If x Is Nothing Then  
                Return -1  
            Else  
                If y Is Nothing Then  
                    Return 1  
                Else  
                    Dim sx As Integer = 0  
                    Dim sy As Integer = 0  
                    Try  
                        ' Ignoring the leading character to have  
numerical sorting.  
                        sx = Integer.Parse(x.ToString().Substring(1))  
                        sy = Integer.Parse(y.ToString().Substring(1))  
                        Return sy - sx  
                    Catch  
                    End Try  
                End If  
            End If  
        End If  
    End Function  
End Class
```

16. Add this code to the **Main** method to use this custom comparer to sort column A.

[C#]

```
// Create an arraylist of random MyObjects.
ArrayList list = new ArrayList();
Random r = new Random();

for(int i = 0; i < 10; i++)
{
    list.Add(new MyObject(r.Next(20)));
}

// Create a Grouping.Engine object.
Engine groupingEngine = new Engine();

// Set its datasource.
groupingEngine.SetSourceList(list);

// Display the data before sorting.
foreach(Record rec in groupingEngine.Table.Records)
{
    MyObject obj = rec.GetData() as MyObject;
    if(obj != null)
    {
        Console.WriteLine(obj);
    }
}

// Pause
Console.ReadLine();

// Custom sort column A.
// Get an IComparer object to handle the custom sorting.
AComparer comparer = new AComparer();
groupingEngine.TableDescriptor.SortedColumns.Add("A");
groupingEngine.TableDescriptor.SortedColumns["A"].Comparer = comparer;

// Display the data before sorting.
foreach(Record rec in groupingEngine.Table.FilteredRecords)
{
    MyObject obj = rec.GetData() as MyObject;
    if(obj != null)
    {
        Console.WriteLine(obj);
    }
}
```

```
// Pause
Console.ReadLine();
```

### [VB.NET]

```
' Create an arraylist of random MyObjects.
Dim list As New ArrayList()
Dim r As New Random()

Dim i As Integer
For i = 0 To 9
    list.Add(New MyObject(r.Next(20)))
Next i

' Create a Grouping.Engine object.
Dim groupingEngine As New Engine()

' Set its datasource.
groupingEngine.SetSourceList(list)

' Display the data before sorting.
Dim rec As Record
For Each rec In groupingEngine.Table.Records
    Dim obj As MyObject = rec.GetData() '
    Dim obj As MyObject = CType(rec.GetData(), MyObject)
    If Not (obj Is Nothing) Then
        Console.WriteLine(obj)
    End If
Next rec

' Pause
Console.ReadLine()

' Custom sort column A.
' Get an IComparer object to handle the custom sorting.
Dim comparer As New AComparer()
groupingEngine.TableDescriptor.SortedColumns.Add("A")
groupingEngine.TableDescriptor.SortedColumns("A").Comparer = comparer

' Display the data before sorting.
Dim rec As Record
For Each rec In groupingEngine.Table.FilteredRecords
    Dim obj As MyObject = CType(rec.GetData(), MyObject)
    If Not (obj Is Nothing) Then
```

```
        Console.WriteLine(obj)
    End If
Next rec

' Pause
Console.ReadLine()
```

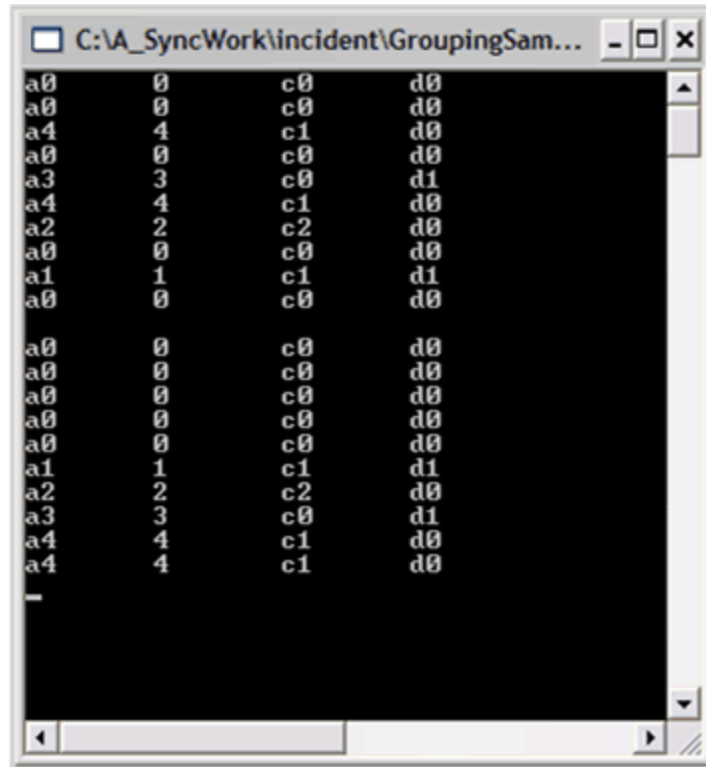


Figure 21: Screen Showing Custom Sorting on Property A

## 4.4 Algebra Supported In Expressions / Filters

Expressions can be any well-formed algebraic combination of the following:

- Property (column) names enclosed with square brackets ([ ])
- Numerical constants and literals
- Algebraic and logical operators

The computations are performed as listed below, with level one operations done first.

- \*, /: multiplication, division
- +, -: addition, subtraction



- <, >, =, <=, >=, <>: less than, greater than, equal, less than or equal, greater than or equal, not equal
- match, like, in, between
- or, and, or



### Note

**1. Alpha constants used with match and like should be enclosed in apostrophes (').**

**2. Logical operators return "1", if the logical expression is True and return "0", if the logical expression is False.**

Given below is some more information on special logical operators:

- **Match**-Returns 1 if there is any occurrence of the right-hand argument in the left-hand argument.

### Example

[CompanyName] match 'RTR' returns 0 for any record whose CompanyName field does not contain RTR anywhere in the string.

- **Like**-Checks if the field starts exactly as specified in the right-hand argument.

### Example

[CompanyName] like 'RTR' returns 1 for any record whose CompanyName field is exactly 'RTR'. You can use an asterisk as a wildcard. [CompanyName] like 'RTR\*' returns 1 for any record whose CompanyName field starts with 'RTR'. [CompanyName] like '\*RTR' returns 1 for any record whose CompanyName field ends with 'RTR'.

- **In**-Checks if the field value is any of the values listed in the right-hand operand. The collection of items used as the right-hand should be separated by commas and enclosed within braces({ }).

### Example

[code] in {1,10,21}, returns 1 for any record whose code field contains 1 or 10 or 21.  
[CompanyName] in {RTR,MAS} returns 1 for any record whose CompanyName field is RTR or MAS.



Note that spaces that are significant in the list, i.e. {RTR,MAS} is not the same as {RTR, MAS}.

- **Between**-Checks if a date field value between the two values is listed in the right-hand operand.

### Example

[date] between {2/25/2004, 3/2/2004} returns 1 for any record whose date field is greater than or equal to 2/25/2004 and less than 3/2/2004. To represent the current date, use the token TODAY. To represent DateTime.MinValue, leave the first argument empty. To represent **DateTime.MaxValue**, leave the second argument empty.

### Custom Functions

Essential Grouping lets you add custom functions to your code that can then be used in expressions.

Following are the limitations on the use of custom functions:

- You cannot use custom functions in algebraic expressions.
- They must be used stand-alone in a simple expression that consists of only the function name and its argument list.
- The argument list can be any number of arguments separated by a delimiter.

## 5 Frequently Asked Questions

---

The following are some common tasks that you might encounter as you use Essential Grouping in your applications:

### 5.1 How to Access the Value of a Record Or Field?

The **groupingEngine.Table.Records** property will allow you to access the records (items) in your IList and particular fields (properties) in the records.

[C#]

```
// Assumes the datasource is an IList holding objects of type MyObject.  
// Retrieves the third object in the list.  
MyObject o = this.groupingEngine.Table.Records[2].GetData() as MyObject;  
  
// A is a public property of MyObject, so "A" is treated as a field.  
object someValue = this.groupingEngine.Table.Records[2].GetValue("A");
```

[VB.NET]

```
' Assumes the datasource is an IList holding objects of type MyObject.  
' Retrieves the third object in the list.  
Dim o As MyObject = CType(Me.groupingEngine.Table.Records(2).GetData(),  
MyObject)  
  
' A is a public property of MyObject, so "A" is treated as a field.  
Dim someValue As Object = Me.groupingEngine.Table.Records(2).GetValue("A")
```



**Note:**

1. The `groupingEngine.Table.Records` collection will give you access to the raw data in your datasource through the `GroupingEngine`.
2. If you have applied filters or have sorted the data and want to access the sorted or filtered data, then you must use the `GroupingEngine.Table.FilteredRecords` collection.
3. This collection reflects the visible state of the data in the `GroupingEngine`.

## 5.2 How to Add Custom Calculations to Expression Fields?

Essential Grouping lets you add your own functions to a function library which can be used in an expression field. In this manner, you can do custom calculations in expressions. This is a two-step process which is given below:

- 1 Register the function name and a delegate with the grouping engine.
- 2 Implement a method that does your calculation.

In the code given below, we add a method named 'Func' that takes two arguments and performs a certain calculation on those arguments.

```
[C#]

// Step 1
// Add function named Func that uses a delegate named ComputeFunc to define a
// custom calculation.
ExpressionFieldEvaluator evaluator =
this.groupingEngine.TableDescriptor.ExpressionFieldEvaluator;
evaluator.AddFunction("Func", new
ExpressionFieldEvaluator.LibraryFunction(ComputeFunc));

// Sample usage in an Expression column.
this.groupingEngine.TableDescriptor.ExpressionFields.Add("test");
this.groupingEngine.TableDescriptor.ExpressionFields["test"].Expression =
"Func([Col1], [Col2])";

//...

// Step 2
// Define ComputeFunc that returns the absolute value of the 1st arg minus 2
// * the 2nd arg.
public string ComputeFunc(string s)
{
    // Get the list delimiter (for en-us, it is a comma).
    char comma =
Convert.ToChar(this.gridGroupingControl1.Culture.TextInfo.ListSeparator);
```

```
string[] ss = s.Split(comma);
if(ss.GetLength(0) != 2)
    throw new ArgumentException("Requires 2 arguments.");

double arg1, arg2;
if(double.TryParse(ss[0], System.Globalization.NumberStyles.Any, null, out
arg1)
&& double.TryParse(ss[1], System.Globalization.NumberStyles.Any, null, out
arg2))
{
    return Math.Abs(arg1 - 2 * arg2).ToString();
}
return "";
}
```

### [VB.NET]

```
' Step 1
' Add function named Func that uses a delegate named ComputeFunc to define a custom
calculation.
Dim evaluator As ExpressionFieldEvaluator =
Me.groupingEngine.TableDescriptor.ExpressionFieldEvaluator
evaluator.AddFunction("Func", New
ExpressionFieldEvaluator.LibraryFunction(ComputeFunc))

' Sample usage in an Expression column.
Me.groupingEngine.TableDescriptor.ExpressionFields.Add("test")
Me.groupingEngine.TableDescriptor.ExpressionFields("test").Expression = "Func([Col1],
[Col2])"

' Step 2
' Define ComputeFunc that returns the absolute value of the 1st arg minus 2 * the 2nd
arg.
Public Function ComputeFunc(ByVal s As String) As String

' Get the list delimiter (for en-us, it is a comma).
Dim comma As Char =
Convert.ToChar(Me.gridGroupingControl1.Culture.TextInfo.ListSeparator)
Dim ss As String() = s.Split(comma)
If ss.GetLength(0) <> 2 Then
Throw New ArgumentException("Requires 2 arguments.")
End If
Dim arg1, arg2 As Double
If Double.TryParse(ss(0), System.Globalization.NumberStyles.Any, Nothing, arg1)
AndAlso _
Double.TryParse(ss(1), System.Globalization.NumberStyles.Any, Nothing, arg2) Then
Return Math.Abs((arg1 - 2 * arg2)).ToString()
End If
Return ""
```

```
' ComputeFunc  
End Function
```

## 5.3 How to Add Expression Fields?

To add an expression field to the data in the Grouping Engine, you must first create an instance of an **ExpressionFieldDescriptor** and add it to the **ExpressionFields** collection in the **TableDescriptor**. The ExpressionFieldDescriptor will allow you to specify a string that holds an algebraic expression using any of the other fields that are in the record.

The following code snippet illustrates this.

**[C#]**

```
// Add an expression property that multiplies field B by 2.1 and adds 3.2  
ExpressionFieldDescriptor efd = new ExpressionFieldDescriptor("MultipleOfB",  
"2.1 * [B] + 3.2");  
this.groupingEngine.TableDescriptor.ExpressionFields.Add(efd);  
  
// Assumes the datasource to be an IList, holding objects of type MyObject.  
MyObject o = this.groupingEngine.Table.Records[2].GetData() as MyObject;  
  
// MultipleOfB is an expression field name and B is a property in MyObject.  
object someValue = this.groupingEngine.Table.Records[2].GetValue("B");  
object someExpressionValue =  
this.groupingEngine.Table.Records[2].GetValue("MultipleOfB");
```

**[VB.NET]**

```
' Add an expression property that multiplies field B by 2.1 and adds 3.2  
Dim efd As New ExpressionFieldDescriptor("MultipleOfB", "2.1 * [B] + 3.2")  
Me.groupingEngine.TableDescriptor.ExpressionFields.Add(efd)  
  
' Assumes the datasource to be an IList, holding objects of type MyObject.  
Dim o As MyObject = CType(Me.groupingEngine.Table.Records(2).GetData(),  
MyObject)  
  
' MultipleOfB is an expression field name and B is a Property in MyObject.  
Dim someValue As Object = Me.groupingEngine.Table.Records(2).GetValue("B")  
Dim someExpressionValue As Object =  
Me.groupingEngine.Table.Records(2).GetValue("MultipleOfB")
```

## 5.4 How to Add Summary Items?

Essential Grouping lets you summarize data by adding **SummaryDescriptor** objects to the schema information stored in the **Engine.TableDescriptor.Summaries** collection. You can have multiple summaries by adding several **SummaryDescriptors**.

[C#]

```
// Create a summary that computes the Int32Aggregate calculations on field B.
SummaryDescriptor sdBInt32Agg = new SummaryDescriptor("BInt32Agg", "B",
SummaryType.Int32Aggregate);

// Add this summary to the Summaries collection.
groupingEngine.TableDescriptor.Summaries.Add(sdBInt32Agg);
```

[VB.NET]

```
' Create a summary that computes the Int32Aggregate calculations on property B.
Dim sdBInt32Agg As New SummaryDescriptor("BInt32Agg", "B",
SummaryType.Int32Aggregate)

' Add this summary to the Summaries collection.
groupingEngine.TableDescriptor.Summaries.Add(sdBInt32Agg)
```



**Note:** There are several overloads of the constructor for the **SummaryDescriptor**. We are using the overload that accepts a **SummaryType** enum as the third argument. This **SummaryType** will allow you to pick out some predefined calculations such as the **Int32Aggregate** functions like **Max**, **Min**, **Sum** and **Average**. There are enums that specify **double**, **boolean**, and other aggregate types. Here we choose **Int32** as that is the type of value we see in the **B** property in the data.

## 5.5 How to Bind a Datasource to the Grouping Engine?

Essential Grouping can use any **IList** object holding objects and a common **System.Type** as its datasource. The public properties of the common type can be used to group, sort and summarize the data in the **IList**.

### Example

The following code shows how to set an **IList** object to be the data source of a **GroupingEngine** object. Within Essential Grouping, the items in your **IList** datasource are referred to as records.

**[C#]**

```
using Syncfusion.Grouping;

// Create a Grouping.Engine object.
Engine groupingEngine = new Engine();

// Set its datasource.
groupingEngine.SetSourceList(list);
```

**[VB.NET]**

```
Imports Syncfusion.Grouping

' Create a Grouping.Engine object.
Dim groupingEngine As New Engine()

' Set its datasource.
groupingEngine.SetSourceList(list)
```

## 5.6 How to Clear a Filter?

To clear all filters, call the `groupingEngine.TableDescriptor.RecordFilters.Clear` method.

**[C#]**

```
// Removes all the filters associated with the table.
this.gridGroupingControl1.TableDescriptor.RecordFilters.Clear();

// Removes the Filter associated by sending the argument as
RecordFilterDescriptor.name
this.gridGroupingControl1.TableDescriptor.RecordFilters.Remove(RecordFilterDe
scriptor.Name);

// Removes the RecordFilter associated by mentioning as index.
this.gridGroupingControl1.TableDescriptor.RecordFilters.RemoveAt();
```

**[VB.NET]**

```
' Removes all the filters associated with the table.
Me.gridGroupingControl1.TableDescriptor.RecordFilters.Clear()

' Removes the Filter associated by sending the argument as
```



```
RecordFilterDescriptor.name
Me.gridGroupingControl1.TableDescriptor.RecordFilters.Remove()

' Removes the RecordFilter associated by mentioning as index.
Me.gridGroupingControl1.TableDescriptor.RecordFilters.RemoveAt()
```



**Note:** To remove a particular filter, use the `groupingEngine.TableDescriptor.RecordFilters.Remove` or `groupingEngine.TableDescriptor.RecordFilters.RemoveAt`. To use `Remove`, you will need a reference to the `RecordFilterDescriptor` object that you want to delete or your `RecordFilterDescriptor` object would have to be named (setting the `RecordFilterDescriptor.Name` property or by passing a name string into its overloaded constructor).

## 5.7 How to Clear a Grouping?

To clear all grouping, call the `groupingEngine.TableDescriptor.GroupedColumns.Clear` method.

[C#]

```
// Removes the grouping of all the grouped columns.
this.gridGroupingControl1.TableDescriptor.GroupedColumns.Clear();

// Removes grouping of the column mentioned as an argument.
this.gridGroupingControl1.TableDescriptor.GroupedColumns.Remove(Name);

// Removes grouping of the column mentioned as column index.
this.gridGroupingControl1.TableDescriptor.GroupedColumns.RemoveAt(index);
```

[VB.NET]

```
' Removes the grouping of all the grouped columns.
Me.gridGroupingControl1.TableDescriptor.GroupedColumns.Clear()

' Removes grouping of the column mentioned as an argument.
Me.gridGroupingControl1.TableDescriptor.GroupedColumns.Remove();

' Removes grouping of the column mentioned as column index.
Me.gridGroupingControl1.TableDescriptor.GroupedColumns.RemoveAt()
```



**Note:** To remove a particular grouping, use `groupingEngine.TableDescriptor.GroupedColumns.Remove` or `groupingEngine.TableDescriptor.SortedColumns.RemoveAt`.

## 5.8 How to Clear a Sort?

To clear all sorts, call the `groupingEngine.TableDescriptor.SortedColumns.Clear` method.

[C#]

```
// Removes all the sorting associated with the table.
this.gridGroupingControl1.TableDescriptor.SortedColumns.Clear();

// Removes the sorting of the column mentioned as argument.
this.gridGroupingControl1.TableDescriptor.SortedColumns.Remove (Name of the
column);

// Removes the sorting of the column mentioned as column index.
this.gridGroupingControl1.TableDescriptor.SortedColumns.RemoveAt (index);
```

[VB.NET]

```
' Removes all the sorting associated.
Me.gridGroupingControl1.TableDescriptor.SortedColumns.Clear()

' Removes the sorting of the column mentioned as argument.
Me.gridGroupingControl1.TableDescriptor.SortedColumns.Remove()

' Removes the sorting of the column mentioned as column index.
Me.gridGroupingControl1.TableDescriptor.SortedColumns.RemoveAt()
```



**Note:** To remove a particular sort, use `groupingEngine.TableDescriptor.SortedColumns.Remove` or `groupingEngine.TableDescriptor.SortedColumns.RemoveAt`.

## 5.9 How to Filter a Collection?

To add a filter condition, add a **RecordFilterDescriptor** to the **Engine.TableDescriptor.RecordFilters** collection. The constructor on the **RecordFilterDescription** takes an expression, "[D] LIKE 'd1'". This expression will be **True** only for those items in the list where the string property D has the value d1. Here are some other valid expressions where B is an integer property:

- "[B] = 5 OR [B] < 0"
- "[D] LIKE 'd1' OR [B] = 2"

[C#]

```
// Filter on [D] = d1
RecordFilterDescriptor rfd = new RecordFilterDescriptor("[D] LIKE 'd1'");
this.groupingEngine.TableDescriptor.RecordFilters.Add(rfd);
```

[VB.NET]

```
' Filter on [D] = d1
Dim rfd As New RecordFilterDescriptor("[D] LIKE 'd1'")
Me.groupingEngine.TableDescriptor.RecordFilters.Add(rfd)
```

## 5.10 How to Group a Collection?

To sort your data, add the name of the property you want to sort to the **Engine.TableDescriptor.GroupedColumns** collection.

[C#]

```
// Group column A.
groupingEngine.TableDescriptor.GroupedColumns.Add("A");
```

[VB.NET]

```
' Group column A.
groupingEngine.TableDescriptor.GroupedColumns.Add("A")
```

## 5.11 How to Retrieve Summary Item Values?

Summaries are calculated on groups of records. The **TopLevelGroup** is the collection of all records in the **IList** data source. If you have added additional grouping through the **groupingEngine.TableDescriptor.GroupedColumns.Add**, then in addition to the top level group, there will be additional groups available to you. Each group will have an associated summary value. So to retrieve a summary value, you need to specify the group associated with the summary.

For example, the code below assumes that you have grouped the table on field "C" and are looking for the summary associated with the group whose records have field "C" equal to the value "c1". It also shows the same summary value for the TopLevelGroup.

### [C#]

```
// To simplify notation, set this using the statement at the top of your code
file.
using ISummary = Syncfusion.Collections.BinaryTree.ITreeTableSummary;

// Now get the Summary value for the TopLevelGroup group.
ISummary groupSummary =
groupingEngine.Table.TopLevelGroup.GetSummary("BInt32Agg");
Int32AggregateSummary int32Summary = (Int32AggregateSummary) groupSummary;

Console.WriteLine("whole table {0}, {1}, {2}", int32Summary.Sum,
int32Summary.Average, int32Summary.Maximum);

// Value for "c1" group.
groupSummary =
groupingEngine.Table.TopLevelGroup.Groups["c1"].GetSummary("BInt32Agg");
int32Summary = (Int32AggregateSummary) groupSummary;

Console.WriteLine("c1-group {0}, {1}, {2}", int32Summary.Sum,
int32Summary.Average, int32Summary.Maximum);
```

### [VB.NET]

```
' To simplify notation, set this using the statement at the top of your code
file.
Imports ISummary = Syncfusion.Collections.BinaryTree.ITreeTableSummary

' Now get the Summary value for the TopLevelGroup group.
Private groupSummary As ISummary =
groupingEngine.Table.TopLevelGroup.GetSummary("BInt32Agg")
Private int32Summary As Int32AggregateSummary = CType(groupSummary,
Int32AggregateSummary)

Console.WriteLine("whole table {0}, {1}, {2}", int32Summary.Sum,
int32Summary.Average, int32Summary.Maximum)

' Value for "c1" group.
Private groupSummary =
groupingEngine.Table.TopLevelGroup.Groups("c1").GetSummary("BInt32Agg")
Private int32Summary = CType(groupSummary, Int32AggregateSummary)
```

```
Console.WriteLine("c1-group {0}, {1}, {2}", int32Summary.Sum,  
int32Summary.Average, int32Summary.Maximum)
```

## 5.12 How to Sort a Collection?

To sort your data, add the name of the property that you want to sort to the **Engine.TableDescriptor.SortedColumns** collection.

**[C#]**

```
// Sort column A.  
groupingEngine.TableDescriptor.SortedColumns.Add("A");
```

**[VB.NET]**

```
' Sort column A.  
groupingEngine.TableDescriptor.SortedColumns.Add("A")
```

## **Index**

### **A**

Accessing a Particular Group 33  
Adding a Summary 37  
Algebra Supported In Expressions / Filters 56  
ASP.NET 20

### **C**

Concepts and Features 22  
Creating an ArrayList of Objects 22  
Creating Platform Application 16  
Custom Sorting 51

### **D**

Data Binding 22  
Data Manipulation 39  
Deploying Essential Grouping 18  
Deployment Requirements 15  
Documentation 7

### **E**

Expressions 45

### **F**

Filters 40  
Frequently Asked Questions 59

### **G**

Getting Started 16  
Grouping a Table 30

### **H**

How to Access the Value of a Record Or Field? 59  
How to Add Custom Calculations to Expression Fields? 60  
How to Add Expression Fields? 62  
How to Add Summary Items? 63  
How to Bind a Datasource to the Grouping Engine? 63

How to Clear a Filter? 64  
How to Clear a Grouping? 65  
How to Clear a Sort? 66  
How to Filter a Collection? 66  
How to Group a Collection? 67  
How to Retrieve Summary Item Values? 67  
How to Sort a Collection? 69

### **I**

Installation 9  
Installation and Deployment 9  
Introduction to Essential Grouping 4  
Iterating Through the Data 28

### **O**

Overview 4

### **P**

Prerequisites and Compatibility 6

### **R**

Retrieving Summary Values for a Particular Group 38

### **S**

Setting a Datasource In the Grouping Engine 27  
Sorting 48

### **T**

The Grouping.TableDescriptor Class 32

### **U**

Using Grouping 30

### **W**

Where to Find Samples? 9  
Windows / WPF 19

