Data Acquisition & Manipulation

1 Connecting to Data

Introduction

.netCHARTING provides the **DataEngine** object which can be used to automatically obtain data from a variety of data sources. The DataEngine returns a **SeriesCollection** object which contains data that is consumed by the chart.

The DataEngine object is instantiated like so:

```
[C#]
DataEngine de = new DataEngine();


[Visual Basic]
Dim de As New DataEngine()
```

Using data sources.

The **DataEngine** object internally connects to any supported data sources when a **ConnectionString** is specified. Supported data sources include

- MS Access
- MS SQL
- Oracle*
- mySQL*
- ODBC*
- Excel
- XML

*Available with .net framework version 2.0 only.

Unsupported data sources can also be utilized by supplying the DataEngine with a data object such as a DataTable.

When connecting to supported data sources data is queried by setting the SqlStatement property of the **DataEngine**.

Examples

There are different methods of connecting to specific data sources. The following section demonstrates how each supported data source can be utilized.

Access Database

```
[C#]
de.ConnectionString = @"Provider=Microsoft.Jet.OLEDB.4.0;user id=;password=
                         ; data source=" + Server.MapPath("db.mdb");
// A connection string shortcut for access databases is also available.
de.ConnectionString = "db.mdb";
de.SqlStatement = @"SELECT OrderDate,Sum(Total) FROM Orders ";


[Visual Basic]
de.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;user id=;password=;"_
                         & "data source=" & Server.MapPath("db.mdb")
' A connection string shortcut for access databases is also available.
de.ConnectionString = "db.mdb"
de.SqlStatement = "SELECT OrderDate,Sum(Total) FROM Orders "
```

SQL Database

```
[C#]
de.ConnectionString = @"server=server name or IP;uid=username;pwd=password;database=databa
de.SqlStatement = @"SELECT OrderDate,Sum(Total) FROM Orders ";


[Visual Basic]
de.ConnectionString = "server=server name or IP;uid=username;pwd=password;database=databas
de.SqlStatement = "SELECT OrderDate,Sum(Total) FROM Orders "
```

Stored Procedure

```
[C#]
de.StoredProcedure = "myProcedure";
de.AddParameter("@STARTDATE","3/10/02 12:00:00 AM",FieldType.Date);


[Visual Basic]
de.StoredProcedure = "myProcedure"
de.AddParameter("@STARTDATE","3/10/02 12:00:00 AM",FieldType.Date)
```

The **AddParameter** method is optionally used in conjunction with the **StoredProcedure** property to specify any number of parameters for the stored procedure. The following parameter types are supported:

- Text
- Date
- Number
- LongNumber
- Currency
- Double

Excel Files

```
[C#]
de.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;data source= " + Server.MapPath("m
                        + ";Extended Properties=\"Excel 8.0;\"";
de.SqlStatement= @"Select Periods,Sales from [Yearly Summary$]";


[Visual Basic]
de.ConnectionString = "Provider=Microsoft.Jet.OLEDB.4.0;data source=" & Server.MapPath("my
                        & ";Extended Properties=\"Excel 8.0;\""
de.SqlStatement= "Select Periods,Sales from [Yearly Summary$]"
```

XML Files

Xml files can be used as data sources by setting the **Data** property of the DataEngine with the xml document file name:

```
[C#]
de.Data = "Orders.xml";


[Visual Basic]
de.Data = "Orders.xml"
```

Other Sources

Firebird and other unsupported sources

.netCHARTING can also parse supported data objects generated from any data source using the respective .net providers. If you can access your data in your .NET application you can chart it with .netCHARTING in a few simple lines of code.

Supported Data Objects

- DataSet
- DataTable
- DataView
- String (Raw xml)
- String (xml file name)
- XmlDocument

These objects are consumed by the DataEngine's **Data** property.

Example:

```
[C#]
de.Data = myDataTable;
```

```
[Visual Basic]
de.Data = myDataTable
```

2 Getting Data

2.1 Using Query Strings

This tutorial describes how database queries can be used to populate chart series and elements.

Simple chart

By default, the first column returned by your query is mapped on the x axis and the second column is mapped onto the y axis.

Consider:
'SELECT name, age FROM Employees'

This query will create a chart where the names are on the x axis, ages on the y axis, and might look something like this:

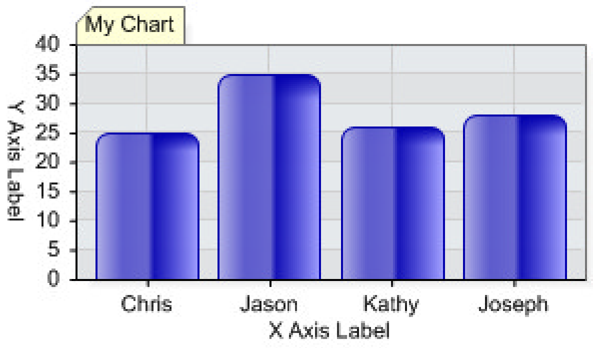


Figure 1. Using a simple query

Multiple series from a single query

Returning a third column splits the data into series. This feature is referred to as 'SplitBy'.

Consider:
'SELECT name, age, division FROM Employees GROUP BY division'

This time 3 columns are returned and rows are grouped by the division column. Because the third column is supplied, two series are created, one for each unique value of the division column. The resulting chart will look something like this:

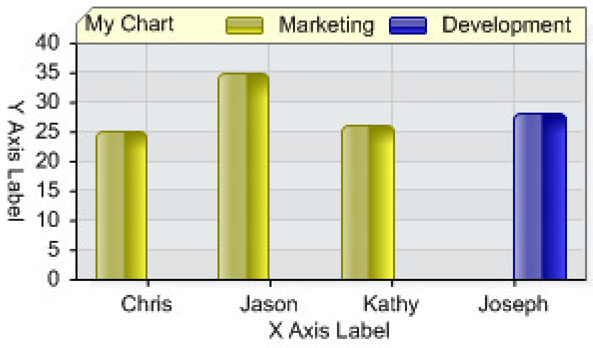


Figure 2. Multiple series using a query
Another characteristic of the splitBy functionality is the aggregation of grouped values. For this example consider a database table with 3 fields [Date, Items_Sold, Sales_Rep] . Each day an entry is added with the number of items sold for each sales rep. In our database there are two and they have been selling from January - May.
Consider:
'SELECT Date, Items_Sold, Sales_Rep FROM Sales GroupBy Sales_Rep'

In this case the sales of each representative are aggregated for each month.
* This sample also uses the dategrouping setting of (TimeInterval.Month), see **dataEngine (Section 5)** for more info.

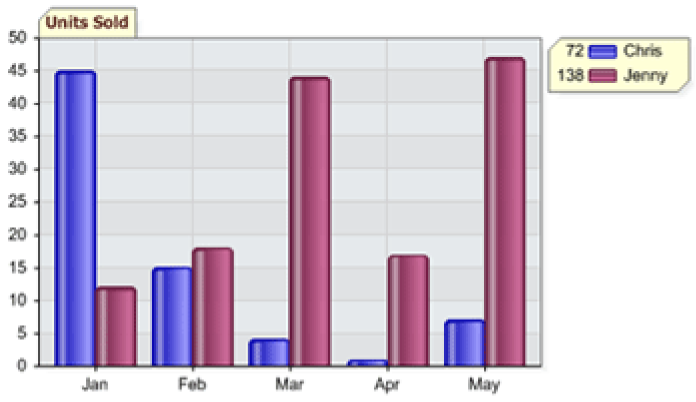


Figure 3. Multiple series with date grouping.

Count occurrences of a fields value.

For this case, let's say we have a table which is a log of logins into your main company server. The table contains a 'date' column and a 'name' column and each row is a login entry. To show the number of logins for each user we can use the following query:

Query:
'SELECT name, 1 FROM Orders GROUP BY name;'

This will create a table with a name and a value (1) for each login. This method takes advantage of the DataEngine's aggregation feature and the second column will be summed.

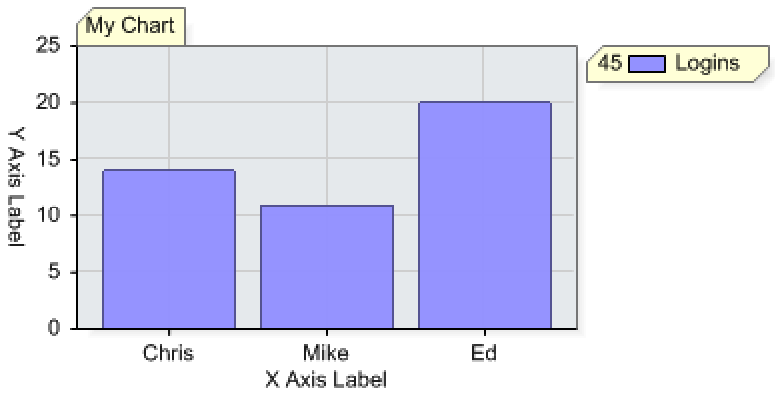


Figure 4. Counting occurences.

2.2 Getting Multiple Series

Using Query Strings (Series based on field)

To get multiple series automatically using the data engine you must specify a third column in the query string. This feature is referred to as 'SplitBy'.

Consider:

'SELECT name, age, division FROM Employees GROUP BY division'

Three columns are returned and rows are grouped by the division column. Because the third column is supplied, two series are created, one for each unique value of the division column. The resulting chart will look something like this:

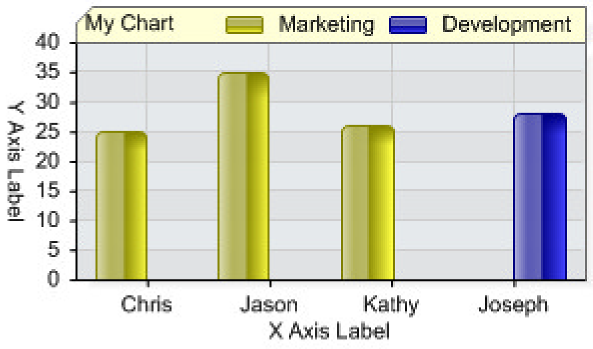


Figure 1: Returning a third column splits the data into series.

Using a single SQL Statement (Series Per Column)

Multiple series can be automatically generated for each column using multiple YValue tokens in the DataFields property.

For example, consider the following code:

```
[C#]
myDataEngine.SQLStatement = "SELECT myDate, GroupA, GroupB, GroupC FROM Stats";
myDataEngine.DataFields = "XValue=myDate,YValue=GroupA,YValue=GroupB,YValue=GroupC";

[Visual Basic]
myDataEngine.SQLStatement = "SELECT myDate, GroupA, GroupB, GroupC FROM Stats"
myDataEngine.DataFields = "XValue=myDate,YValue=GroupA,YValue=GroupB,YValue=GroupC"
```

Assigning Series Names

This will generate three series, all of which will use the myDate column for the x values and their individual groupX columns for y values. Custom series names can also be specified in the data fields. The following code does the same as the above but will assign custom series names.

```
[C#]
myDataEngine.SQLStatement = "SELECT myDate, GroupA, GroupB, GroupC FROM Stats";
myDataEngine.DataFields = "XValue=myDate,YValue=GroupA=Group Alpha,YValue=GroupB=Group Bet

[Visual Basic]
myDataEngine.SQLStatement = "SELECT myDate, GroupA, GroupB, GroupC FROM Stats"
myDataEngine.DataFields = "XValue=myDate,YValue=GroupA=Group Alpha,YValue=GroupB=Group Bet
```

Using Data Fields (Series Per Column)

If you are getting many columns from the database and would like to specify which column to use as the splitBy column you can do so using the DataFields property.

```
[C#]
myDataEngine.DataFields = "Name=name,YValue=age,SplitBy=Division";
[Visual Basic]
myDataEngine.DataFields = "Name=name,YValue=age,SplitBy=Division"
```

Query Database Multiple Times

Another way to do this, is to query your database separately for each series.

```
[C#]
DataEngine de = new DataEngine(connectionString);
de.SqlStatement = "SELECT name, dataOne FROM table";
Chart.SeriesCollection.Add(de.GetSeries());
de.SqlStatement = "SELECT name, dataTwo FROM table";
Chart.SeriesCollection.Add(de.GetSeries());


[Visual Basic]
Dim de As New DataEngine(connectionString)
```

```
de.SqlStatement = "SELECT name, dataOne FROM table"
Chart.SeriesCollection.Add(de.GetSeries())
de.SqlStatement = "SELECT name, dataTwo FROM table"
Chart.SeriesCollection.Add(de.GetSeries())
```

Avoid Querying Multiple Times

If querying the database multiple times is not desirable the same can be accomplished by using a DataTable.

```
[C#]
DataEngine de = new DataEngine();
DataTable dt = new DataTable();
// Populate the datatable from your database.
dt = (...);
de.Data = dt;
de.DataFields = "YAxis=name,XAxis=DataOne";
Chart.SeriesCollection.Add(de.GetSeries());
de.DataFields = "YAxis=name,XAxis=DataTwo";
Chart.SeriesCollection.Add(de.GetSeries());
```

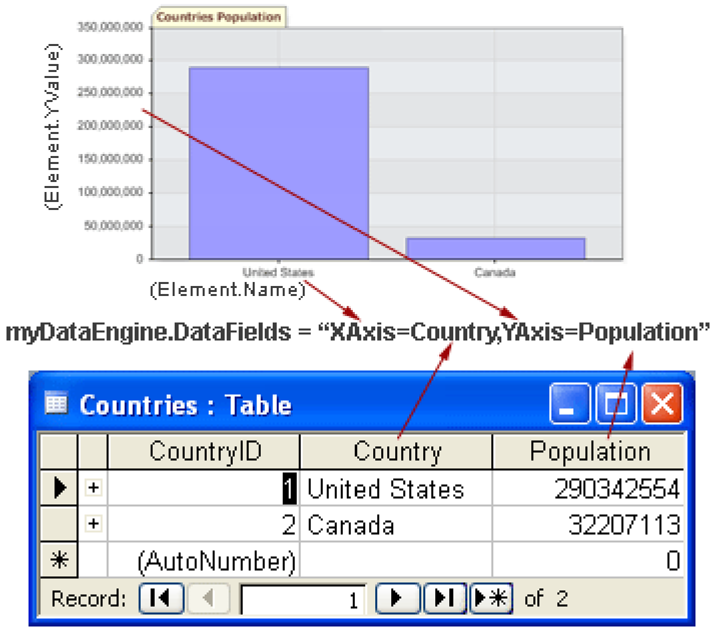
```
[Visual Basic]
Dim de As New DataEngine()
Dim dt As New DataTable()
' Populate the datatable from your database.
dt = (...)
de.Data = dt
de.DataFields = "YAxis=name,XAxis=DataOne"
Chart.SeriesCollection.Add(de.GetSeries())
de.DataFields = "YAxis=name,XAxis=DataTwo"
Chart.SeriesCollection.Add(de.GetSeries())
```

3 Data Field Mapping

Introduction

Data fields are a very useful tool when translating your database structure to chart data. It allows you to map which column of your data table or query populates a given element property.

See Illustration



```
myDataEngine.DataFields = "XAxis=Country,YAxis=Population"
```

CountryID	Country	Population
1	United States	290342554
2	Canada	32207113
(AutoNumber)		0

For example, if you have a column which represents certain information you want to use as a tool tip, it can easily be mapped using data fields. The 'DataFields' property of DataEngine and Chart.Series is a string which can be supplied as a comma delimited list of element property, to data column, relationships.

For example, in the following DataFields string

`"YValue=Cost,Name=Brand,ToolTip=Description"`; Cost, Brand, and Description are all columns in a data table. YValue, Name, and ToolTip are element values.

Custom tokens (Section 4) can also be used to populate elements with additional database columns which can be used by element tool tips, urls, within labels and so on.

The following table outlines the available element values and shortcuts which can be mapped to your data columns.

Token Name	**Description**
Name	Sets Element.Name
Yvalue	Sets Element.YValue or Element.YDateTime depending on the column data type.
YValueStart	Sets Element.XValue or Element.XDateTime depending on the column data type.
Xvalue	Sets Element.XValue or Element.XDateTime depending on the column data type.
XValueStart	Sets Element.XValueStart or Element.XDateTimeStart depending on the column data type.
BubbleSize	Sets Element.BubbleSize
Complete	Sets Element.Complete
Open	Sets Element.Open
Close	Sets Element.Close
High	Sets Element.High
Low	Sets Element.Low
Volume	Sets Element.Volume
tooltip	Sets Element.ToolTip
urltarget	Sets Element.URLTarget
Url	Sets Element.URL
LabelTemplate	Sets Element..LabelTample
	Specifies the name of the table to use when multiple tables exist, for example when

Table	a DataSet is used.

Shortcuts and special functionality tokens

Token Name	**Description**
Bubble	Sets Element.BubbleSize
price	Sets Element.Close
Ganttcomplete	Sets Element.Complete
Ganttname	Sets Element.Name
Ganttenddate	Sets Element.YDateTime
Ganttstartdate	Sets Element.YDateTimeStart
Ganttend	Sets Element.YValue or Element.YDateTime if DateTime data type.
Ganttstart	Sets Element.YValueStart or Element.YStartDateTime if DateTime data type.
Splitby	Sets Element.SplitBy (Special)
Yaxis	Sets the element values associated with the y axis of the chart in use. *ChartType.ComboHorizaontal* sets Element.Name. *All Others* set Element.Yvalue, if column type is 'DateTime' Element.YDateTime will be set instead.
Xaxis	Sets the element values associated with the x axis of the chart in use. *ChartType.ComboHorizontal* sets Element.Yvalue, if column type is 'DateTime' Element.YDateTime will be set instead.. *ChartType.Scatter* sets Element.Xvalue, if column type is 'DateTime' Element.XDateTime will be set instead. *All Other* sets Element.Name. and/or if date it sets Element.XDateTime.

Data fields and XML documents

Xml Documents dont store data exactly as databases do.

Using Data Fields

"[Token]=[Data Column],..."

Escape Character
Some database columns can have characters such as commas ','. In order to reference these column names an escape character can be used.
"YAxis=Tom\,Harry"

Sample Data

This data will be used to demonstrate how different chart types can be achieved by mapping the appropriate element values.

Table 1	**Column**	**(string) Name**	**(double) StartValue**	**(double) EndValue**	**(double) Completed**		
	Sample Data	Chris	3	15	25		
Table 2	**Column**	**(string) Name**	**(DateTime) StartValue**	**(DateTime) EndValue**	**(double) Completed**		
	Sample Data	Chris	3/5/2003	8/10/2003	25		
Table 3	**Column**	**(string) Name**	**(double) Cost**	**(double) TopSpeed**	**(double) HorsePower**		
	Sample Data	Chevy	22,000	120	185		
Table 4	**Column**	**(string) Stock**	**(double) Close**	**(double) Open**	**(double) Low**	**(double) High**	**(DateTime) Date**
	Sample Data	MSFT	176	120	100	185	11/12/2003

The above tables are color coded for reference to which table the column name refers. If there is more than one example in sub sections such as 'Simple Graph', they are equivalent.

ChartType.Combo & ChartType.ComboSideBySide

Default DataFields	Defaults are chosen based on the number of columns returned by the query. "xAxis=(1stColumn),yAxis=(2stColumn)" "xAxis=(1stColumn),yAxis=(2stColumn),splitBy=(3rdColumn)"
Applicable tokens	**Element Data Properties** YValue, YValueStart, Name, Complete, Tooltip, UrlTarge, Url **Shortcuts** xAxis, yAxis, SplitBy **Gantt Shortcuts**

GanttName, GanttStart, GanttEnd, GanttComplete

Examples

Simple Graph	"xAxis=Name,yAxis= EndValue" "Name=Name,yValue=EndValue"
Range (Gantt) Columns	"xAxis=Name,yValue=EndValue,yValueStart=StartValue"
Range columns /w Complete indicator	"xAxis=Name,yValue=EndValue,yValueStart=StartValue ,Complete=Completed"
Date Range columns /w complete indicator.	"xAxis=Name,yValue=EndValue,yValueStart=StartValue ,Complete=Completed"

NOTE: For this chart type "xAxis=Name" is the same as "Name=Name".

ChartType.Horizontal

Default DataFields	Defaults are chosen based on the number of columns returned by the query. "xAxis=(1stColumn),yAxis=(2stColumn)" "xAxis=(1stColumn),yAxis=(2stColumn),splitBy=(3rdColumn)"
Applicable tokens	**Element Data Properties** YValue, YValueStart, Name, Complete, Tooltip, UrlTarge, Url **Shortcuts** xAxis, yAxis, SplitBy **Gantt Shortcuts** GanttName, GanttStart, GanttEnd, GanttComplete

Examples

Simple Graph	"xAxis=Name,yAxis= EndValue" "Name=Name,yValue=EndValue"
Range (Gantt) Columns	"xAxis=Name,yValue=EndValue,yValueStart=StartValue"
Range columns /w Complete indicator	"xAxis=Name,yValue=EndValue,yValueStart=StartValue ,Complete=Completed"
Date Range columns /w complete indicator.	"xAxis=Name,yValue=EndValue,yValueStart=StartValue ,Complete=Completed"

NOTE: For this chart type "xAxis=Name" is the same as "Name=Name".

ChartType.Gantt

Default DataFields	Defaults are chosen based on the number of columns returned by the query. "yValueStart=(1stColumn),yValue=(2stColumn)" "yValueStart=(1stColumn),yValue=(2stColumn),Name=(3rdColumn)" "yValueStart=(1stColumn),yValue=(2ndColumn),Name=(3rdColumn),Complete=(4thColumn)"
Applicable tokens	**Element Data Properties** YValue, YValueStart, Name, Complete, Tooltip, UrlTarge, Url **Shortcuts** xAxis, yAxis, SplitBy **Gantt Shortcuts** GanttName, GanttStart, GanttEnd, GanttComplete

Examples

Range (Gantt) Bars	"yValueStart=StartValue,yValue=EndValue"
Range Columns /w Complete	"yValueStart=StartValue,yValue=EndValue, Name=Name,Complete=Completed"
Date Range Columns /w Complete	"yValueStart=StartValue,yValue=EndValue,Name=Name,Complete=Completed"

ChartType.Radar & ChartType.Pie

Default DataFields	Defaults are chosen based on the number of columns returned by the query. "xAxis=(1stColumn),yAxis=(2stColumn)" "xAxis=(1stColumn),yAxis=(2stColumn),splitBy=(3rdColumn)"

Applicable tokens	**Element Data Properties** YValue, Name, Tooltip, UrlTarge, Url **Shortcuts** xAxis, yAxis, SplitBy

Examples

Simple Chart	"yAxis=EndValue,xAxis=Name" ***Note:*** Though there are no x and y axes in a pie chart the fields are mapped like so: "yValue=EndValue,Name=Name" ***Note:*** The radar graph has both x and y axes.

ChartType.Scatter

Default DataFields	Defaults are chosen based on the number of columns returned by the query. "xAxis=(1stColumn),yAxis=(2stColumn)" "xAxis=(1stColumn),yAxis=(2stColumn),splitBy=(3rdColumn)"
Applicable tokens	**Element Data Properties** YValue, XValue, Name, Complete, Tooltip, UrlTarge, Url **Shortcuts** xAxis, yAxis, SplitBy

Examples

Simple Scatter Chart	"xAxis=HorsePower,yAxis=TopSpeed" "xValue=HorsePower,yValue=TopSpeed"

ChartType.Bubble

Default DataFields	Defaults are chosen based on the number of columns returned by the query. "xAxis=(1stColumn),yAxis=(2stColumn)" "xAxis=(1stColumn),yAxis=(2stColumn),BubbleSize=(3rdColumn)"
Applicable tokens	**Element Data Properties** YValue, XValue, BubbleSize, Name, Tooltip, UrlTarge, Url **Shortcuts** xAxis, yAxis, Bubble

Examples

Simple Scatter Chart	"xAxis=HorsePower,yAxis=TopSpeed,BubbleSize=Cost" "xValue=HorsePower,yValue=TopSpeed,Bubble=Cost"

ChartType.Financial

Default DataFields	Defaults are chosen based on the number of columns returned by the query. "xAxis=(1stColumn),yAxis=(2stColumn)" "xAxis=(1stColumn),yAxis=(2stColumn),Volume=(3rdColumn)" "xAxis=(1stColumn),yAxis=(2stColumn),Volume=(3rdColumn),SplitBy=(3thColumn)"
Applicable tokens	**Element Data Properties** Open, Close, High, Low, Volume, Name, Complete, Tooltip, UrlTarge, Url **Shortcuts** xAxis, yAxis, SplitBy, Price

Examples

HLC Chart	"xAxis=Date,High=High,Low=Low,Price=Close"

4 Custom Data Attributes

Custom element attributes.

Additional information stored in databases can be extracted and used in a chart to provide further information in a tool tip, an element's label, within the element's hot spot URL, and any other related text strings.

Extracting from a database

Extracting attributes from a database can be accomplished with a single line of code using the **DataFields** property of a **DataEngine** object..

Assume the following database columns:

- EmployeeID
- FullName
- Department
- EmailAddress
- PhoneNumber
- AveragePerformance

Using this database we will create a chart that shows each employee's name on the x axis and they're average performance on the y axis. When we mouse over a column in the chart we want to see the employee's ID, department, and phone number. When we click a given column we want to send the employee an email.

The first step is to specify the DataFields property.

```
[C#]
DataEngine de = new DataEngine();
de.ConnectionString = "...";
de.SqlStatement =  "SELECT * FROM
myTable";  de.DataFields="xAxis=FullName,yAxis=AveragePerformance,id=EmployeeID,"
 + "department=Department,email=EmailAddress,phone=PhoneNumber";
[Visual Basic]
Dim de As New DataEngine()
de.ConnectionString = "..."
de.SqlStatement =  "SELECT * FROM
myTable"  de.DataFields="xAxis=FullName,yAxis=AveragePerformance,id=EmployeeID,"_
 & "department=Department,email=EmailAddress,phone=PhoneNumber"
```

The next step is to specify a template for the element's tooltip and url.

```
[C#]
Chart.DefaultSeries.DefaultElement.ToolTip = "ID: %id \n Department: %department \n Phone
Chart.DefaultSeries.DefaultElement.URL = "mailto:%email";


[Visual Basic]
Chart.DefaultSeries.DefaultElement.ToolTip = "ID: %id " & vbCrLf &
" Department: %department " & vbCrLf & " Phone Number: %phone"
Chart.DefaultSeries.DefaultElement.URL = "mailto:%email"
```

This complex and highly functional chart is now ready.

Adding manually

Element attributes can be populated manually using the following method.

```
[C#]
Element e = new Element();
e.Name = "myElement";
e.YValue = 15;
e.CustomAttributes["Phone"] = "555-8593";
e.CustomAttributes["Department"] = "Marketing";
[Visual Basic]
Dim e As New Element()
e.Name = "myElement"
e.YValue = 15
e.CustomAttributes("Phone") = "555-8593"
```

```
e.CustomAttributes("Department") = "Marketing"
```

Series Level attributes

Attributes can be added at the series level to specify information common to all elements. These can be used in legend entry labels, legend entry tool tips and URLs.

Let's generate a SeriesCollection using the data engine object from the above example but we will modify it to only extract elements from a specific department.

```
[C#]
de.SqlStatement = "SELECT * FROM myTable WHERE department = "marketing";
SeriesCollection sc = de.GetSeries();
// The above generated 1 series with a number of employee elements.
// Now we can add some attributes for this specific to the marketing department.
sc[0].Name = "Marketing";
sc[0].DefaultElement.CustomAttributes["DepartmentPhoneNumber"] = "555-2414";
sc[0].DefaultElement.CustomAttributes["DepartmentManagerName"] = "555-2414";
sc[0].DefaultElement.CustomAttributes["DepartmentAddress"] = "555-2414";
[Visual Basic]
de.SqlStatement = "SELECT * FROM myTable WHERE department = "marketing"
Dim sc As SeriesCollection
sc = de.GetSeries()
' The above generated 1 series with a number of employee elements.
' Now we can add some attributes for this specific to the marketing department.
sc(0).Name = "Marketing"
sc(0).DefaultElement.CustomAttributes("DepartmentPhoneNumber") = "555-2414"
sc(0).DefaultElement.CustomAttributes("DepartmentManagerName") = "555-2414"
sc(0).DefaultElement.CustomAttributes("DepartmentAddress") = "555-2414"
```

The series now has additional information stored. Let's use it in the series' legend entry tool tip.

```
[C#]
Chart.DefaultSeries.LegendEntry.ToolTip = "Manager: %DepartmentManager \n "
                   + "Phone: %DepartmentPhoneNumber \n Address: %DepartmentAddress";
// Elements can also use the series level attributes
Chart.DefaultSeries.DefaultElement.ToolTip = "%Name \n Phone #: %phone \n "
 + "Department Phone #: %DepartmentPhoneNumber";


[Visual Basic]
Chart.DefaultSeries.LegendEntry.ToolTip = "Manager: %DepartmentManager \r "_
                   & "Phone: %DepartmentPhoneNumber " & vbCrLf & " Address: %DepartmentAdd
' Elements can also use the series level attributes
Chart.DefaultSeries.DefaultElement.ToolTip = "%Name " & vbCrLf & " Phone #:  %phone "_
 & vbCrLf & " " & "Department Phone #: %DepartmentPhoneNumber"
```

The above may tool tips may look like this:

John Doe
Phone #: 555-5426
Department Phone #: 555-1349

Sample: customAttributes.aspx

5 Data Engine

Data Engine

The DataEngine object connects to databases or consumes data objects such as a 'DataTable' and converts them to a SeriesCollection which can then be manipulated and used to generate a chart. The DataEngine offers many data manipulation features saving countless hours of development, particularly for date specific aggregation.

We encourage you to review the following tutorials before working with the DataEngine:

- **Connecting to Data (Section 1)**
- **Using DataFields (Section 3)**

The basic concept of using the data engine is to specify a database, connection string and retrieve the SeriesCollection it generates. A basic example:

```
[C#]
DataEngine de = new DataEngine(connectionString,queryString);
SeriesCollection mySC = de.GetSeries();


[Visual Basic]
Dim de As New DataEngine(connectionString,queryString)
SeriesCollection mySC = de.GetSeries()
```

Filtering out date ranges

When specifying a start and end date in a database query, the following method should be used.

```
[C#]
DataEngine de = new DataEngine();     // Instantiate the data engine object
de.StartDate = new DateTime(2002,1,1,8,0,0);     // Specify the start date
de.EndDate = new DateTime(2002,1,1,23,59,59);      // Specify the end date.
//Specify a query.
de.SqlStatement = "SELECT names, values FROM myTable WHERE start > #StartDate# AND end < #


[Visual Basic]
Dim de As New DataEngine()     ' Instantiate the data engine object
de.StartDate = New DateTime(2002,1,1,8,0,0)     ' Specify the start date
de.EndDate = New DateTime(2002,1,1,23,59,59)      ' Specify the end date.
'Specify a query.
de.SqlStatement = "SELECT names, values FROM myTable WHERE start > #StartDate# AND end < #
```

Notes:

- The #StartDate# and #EndDate# tokens can be used in SQL statements in which case the dates set for de.StartDate and de.EndDate will replace the tokens before the SqlStatement is executed.
- StartDate and EndDate properties affects the dates listed in the title when Chart.ShowDateInTitle is used.
- All date values in your SQL statements should be wrapped with the pound symbol: #10/25/2002#, regardless if Access or SQL server is used.
- See also **DataEngine.SqlStatement** | **DataEngine.StartDate** | **DataEngine.EndDate**

Date grouping

This feature controls how the values of a given series are grouped by date. In order to use this option the first column returned by the SqlStatement must be a date/time data type.

Example 1

```
[C#]
de.SqlStatement = "SELECT time, unitsSold FROM sales";
de.DateGrouping = TimeInterval.Days;


[Visual Basic]
de.SqlStatement = "SELECT time, unitsSold FROM sales"
de.DateGrouping = TimeInterval.Days
```

The above will create an aggregated element for each day within the start and end date of your data.

Similar options include:

- Minutes
- Hours
- Days
- Weeks
- Months
- Quarters
- Years

Example 2

```
[C#]
myDataEngine.SqlStatement = "SELECT time, unitsSold FROM sales";
myDataEngine.DateGrouping = TimeInterval.Day;

[Visual Basic]
myDataEngine.SqlStatement = "SELECT time, unitsSold FROM sales"
myDataEngine.DateGrouping = TimeInterval.Day
```

When using day instead of days, data is grouped into 24 elements representing each hour of the day. For example, if the date range spans a week the element representing 11pm will contain the sum of all values that fall into that hour throughout the week.

Options include:

- Hour (60 Minutes)
- Day (24 Hours)
- Week (7 Days)
- Month (31 Days)
- Quarter (3 Months)
- Year (12 Months)

See also **DataEngine.DateGrouping** | **TimeInterval**

Limiting Data

Generated data can be limited in two ways. First, you can limit the number of elements returned for each series by using the 'Limit' property of the data engine.

```
[C#]
myDataEngine.Limit = "5";

[Visual Basic]
myDataEngine.Limit = "5"
```

Notes

- This property is a string not a numeric value.
- If DateGrouping is used, limit has no effect, you can limit the return with the StartDate and EndDate in such a case.
- When data is limited, elements with the lowest y value are eliminated first.

The second way is to limit the number of series generated when using split by. For an example of SplitBy, see: Tutorials > Simple Queries > Multiple Series from a Single Query.

```
[C#]
myDataEngine.SplitByLimit = "2";

[Visual Basic]
myDataEngine.SplitByLimit = "2"
```

Notes

- This property is a string not a numeric value.
- *SplitBy Defined* - SplitBy occurs automatically based on the values returned by the SqlStatement property. When a 3rd column is returned from your SQL statement it automatically creates any number of new series based on the value provided. This value must be returned as a field in the SQL

statement defined for the series. For example if you have a number of sales by customer, you could choose to graph those values by week in which case you would have 1 series with 4 values for a month or, using SplitBy, you could choose to split by customer and see the individual breakdown of sales by customer - with a separate series for each customer! See the SqlStatement property for more information.

- Series with elements whose summed y values are the lowest are eliminated first.
- See also **DataEngine.Limit** | **DataEngine.SplitByLimit**

Show data eliminated with Limit properties

The additional series not shown due to the use of SplitByLimit, or the additional elements not shown due to the use of Limit are aggregated into a single series or element respectively, and graphed alongside the main data when the ShowOther property is true.

```
[C#]
myDataEngine.ShowOther = true;
myDataEngine.OtherElementText = "The Rest";


[Visual Basic]
myDataEngine.ShowOther = True
myDataEngine.OtherElementText = "The Rest"
```

Notes:

- The 'OtherElementText' property will be used as the name of the aggregated element or as the name of the aggregated series.
- See also **DataEngine.ShowOther** | **DataEngine.OtherElementLabel**

Get data eliminated by Limit properties

If you would like to show series eliminated by Limit or SplitByLimit when drilling down or in a legend box, the LimitMode enumeration can be used. For example if you limit data to 5 and would like to see the rest, the LimitMode.ExcludeTop enumeration member can be used.

```
[C#]
myDataEngine.LimitMode = LimitMode.ExcludeTop;
myDataEngine.Limit = "5";


[Visual Basic]
myDataEngine.LimitMode = LimitMode.ExcludeTop
myDataEngine.Limit = "5"
```

Focus Limit on a specific Series

This feature allows limit to be bound to a specific series. First the series will be limited based on the specified value, then .netCHARTING will automatically match any remaining series to that limit order rather than limiting for each series independently.

```
[C#]
myDataEngine.Limit = "5";
myDataEngine.LimitPrimarySeries = "customers";


[Visual Basic]
myDataEngine.Limit = "5"
myDataEngine.LimitPrimarySeries = "customers"
```

Formatting

The data engine may populate an element's name property. The name property is a string, therefore, in order to ensure proper formatting of those values we can set the 'FormatString' and 'CultureName' properties of the data engine:

```
[C#]
myDataEngine.FormatString = "d";
myDataEngine.CultureName = "en-US";


[Visual Basic]
myDataEngine.FormatString = "d"
myDataEngine.CultureName = "en-US"
```

Notes:

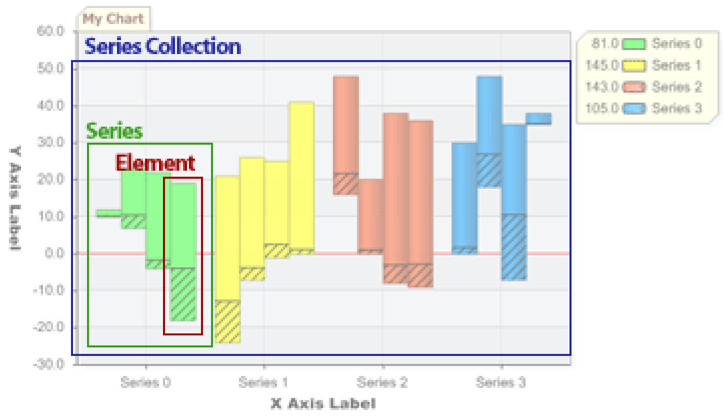
- "en-US" is the default culture name.
- See also **DataEngine.FormatString** | **DataEngine.CultureName**

6 Data Model

.netCHARTING Data Classes

The .netCHARTING data model consists of elements which collectively become a series which can be added to a SeriesCollection.

See Illustration



Data Class	Description
Element	Represents an element on the chart and facilitates the following. ● Provides properties to hold different types of data such as y value, x value, % completed for Gantt charts and many more. Below Element.YValue is demonstrated. ● Controls all other variables concerning a single element such as it's label color, marker type etc.
Series	Utilizes the ElementCollection class to hold a set of elements and facilitates the following. ● Provides a mechanism for setting default properties of all elements within it. ● Controls all series specific properties such as it's LegendEntry or 2D line dash style (if applicable). ● Derives an element based on a **Calculation** which is performed on the elements within it. ● Provides data manipulation methods such as sorting and trimming of it's elements.
SeriesCollection	Contains a collection of series and facilitates the following. ● Derives series based on a **Calculation** which is performed on all the series within it. ● Provides data manipulation methods such as Transpose.
ElementCollection	Used by the Series class to hold a collection of elements.

Using Data Classes

The following example creates a **SeriesCollection** containing one **Series** which contains a single **Element** using the following steps.

1. Instantiates an element and sets it's properties.
2. Instantiates a series and adds the element to it.
3. Instantiates a SeriesCollection and adds the series to it.

```
[C#]
Element e = new Element();     // Instantiate the element
e.Name = "Element 1";       // Name it
```

```
e.YValue = 10;             // Set the y value
Series s = new Series();     // Instantiate the series
s.Name = "Series 1";       // Name it
s.Elements.Add(e);         // Add element e to series s.
SeriesCollection sc = new SeriesCollection(); // Instantiate a series collection
sc.Add(s);                 // Add series s to the series collection sc.

//The sc object now contains data we can chart which we add here.

Chart.SeriesCollection.Add(sc);
```

```
[Visual Basic]
Dim e As New Element()     ' Instantiate the element
e.Name = "Element 1"       ' Name it
e.YValue = 10              ' Set the y value
Dim s As New Series()      ' Instantiate the series
s.Name = "Series 1"        ' Name it
s.Elements.Add(e)          ' Add element e to series s.
Dim sc As New SeriesCollection() ' Instantiate a series collection
sc.Add(s)                  ' Add series s to the series collection sc.

'The sc object now contains data we can chart which we add here.

Chart.SeriesCollection.Add(sc)
```

Shortcuts

.netCHARTING provides many shortcuts which can make adding data easier. The previous example can also be achieved using the following shortcuts.

Using Chart.Series shortcut.

```
[C#]
Element e = new Element();
e.Name = "Element 1";
e.YValue = 10;
Chart.Series.Add(e);
Chart.SeriesCollection.Add(); // Not passing parameters will add the contents of Chart.Ser
```

```
[Visual Basic]
Dim e As New Element()
e.Name = "Element 1"
e.YValue = 10
Chart.Series.Add(e)
Chart.SeriesCollection.Add() ' Not passing parameters will add the contents of Chart.Serie
```

Using the Chart.Element shortcut.

```
[C#]
Chart.Series.Element.YValue = 10;
Chart.Series.Element.Name = "Element 1";
Chart.Series.Elements.Add();
Chart.SeriesCollection.Add();
```

```
[Visual Basic]
Chart.Series.Element.YValue = 10
Chart.Series.Element.Name = "Element 1"
Chart.Series.Elements.Add()
Chart.SeriesCollection.Add()
```

Using all the shortcuts allows us to reduce the amount of lines necessary to achieve the same result from 8 to 4. The down side is that after the data is added using Add(), it cannot be retrieved for further manipulation.

Take it to the next step

It is still possible to reduce the code even further using **element constructors (C:\inetpub\wwwroot\dotnetCHARTING.ElementConstructor.html)**.

```
[C#]
Chart.SeriesCollection.Add(new Element("Element1",10));
```

```
[Visual Basic]
Chart.SeriesCollection.Add(New Element("Element1",10))
```

Now we took it from 8 lines to 1.

For more information on code shortcuts, see the **Efficient Code ('Shortcuts and Efficient Code' in the on-line documentation)** tutorial.

7 Data Manipulation

⚠ The features discussed in this section require that series objects are populated with elements. This does not happen automatically if the DataEngine is not used. Please see **this kb (http://dotnetcharting.com/kb/article.aspx?id=10392)** for more information.

.netCHARTING facilitates numerous calculations that can be performed on your data. These calculations can be used to

- Derive an element from a series
- Derive a series from a SeriesCollection
- The results can also be used in text reports or as further chart data.

Series.Calculate

First we will construct a simple series with some data we can manipulate.

```
[C#]
Series s = new Series();
s.Elements.Add(new Element("e 1",5));
s.Elements.Add(new Element("e 2",15));
s.Elements.Add(new Element("e 3",23));
s.Elements.Add(new Element("e 4",13));
s.Elements.Add(new Element("e 5",34));


[Visual Basic]
Dim s As New Series();
s.Elements.Add(new Element("e 1",5))
s.Elements.Add(new Element("e 2",15))
s.Elements.Add(new Element("e 3",23))
s.Elements.Add(new Element("e 4",13))
s.Elements.Add(new Element("e 5",34))
```

With our series populated with data, we can derive calculations using the Series.Calculate method which returns a single element.

```
[C#]
Element e = s.Calculate("sum",Calculation.Sum);


[Visual Basic]
Dim e As Element = s.Calculate("sum",Calculation.Sum)
```

The element e now has yValue of 91 which is the sum of all the element yValues in series s. If the elements of series s had other values such as BubbleSize, the resulting e.BubbleSize would contain the sum of those as well.

Series.Sort

Elements within a series can be sorted by any value in ascending or descending order.

```
[C#]
// Sort the series by name
s.Sort(ElementValue.Name, "DESC");
// Sort the series by YValue
s.Sort(ElementValue.YValue, "DESC");


[Visual Basic]
' Sort the series by name
s.Sort(ElementValue.Name, "DESC")
' Sort the series by YValue
s.Sort(ElementValue.YValue, "DESC")
```

Series.Trim

Elements with values that are within a given range can be trimmed form a series using the Series.Trim method. The trim method also

```
[C#]
```

```
// Trim elements with y values between 15 and 20.
s.Trim(ElementValue.YValue, 15,20);


[Visual Basic]
' Trim elements with y values between 15 and 20.
s.Trim(ElementValue.YValue, 15,20)
```

Samples

- seriesSort.aspx
- calendarPattern.aspx

The above example trimmed out elements within the specified range. This method also offers an option to trim outside the specified range using a Boolean parameter as in this code:

```
[C#]
// Trim elements with y values outside 15 and 20.
s.Trim(ElementValue.YValue, 15, 20, true);


[Visual Basic]
' Trim elements with y values outside 15 and 20.
s.Trim(ElementValue.YValue, 15, 20, true)
```

Series and SeriesCollection also have a **SelectiveElementDefaults** method which lets you specify a range of elements in the same way however, it applies specified settings to those elements automatically. SelectiveElementDefaults are discussed in more detail in the **Shortcuts and Efficient Code (on-line documentation)** section.

SeriesCollection.Transpose()

The transpose method manipulates the series and elements of a series collection by making the series names, element names and the element names into series names. This is similar to transposing data in Excel. Basically when the data is laid out in columns and rows, the rows become columns and columns become rows. Changing a ChartType from Combo to ComboSideBySide can achieve the same result as using Combo and transposing the series collection.

SeriesCollection.Sort

Series in a SeriesCollection object can be sorted based on their respective element's values.

```
[C#]
// Consider a SeriesCollection sc that has multiple series, each with several elements.
// Sort the series by name.
sc.Sort(ElementValue.Name,"ASC");
// Sort the series by the sum of their element's y value.
sc.Sort(ElementValue.YValue,"ASC");


[Visual Basic]
' Consider a SeriesCollection sc that has multiple series, each with several elements.
' Sort the series by name.
sc.Sort(ElementValue.Name,"ASC")
' Sort the series by the sum of their element's y value.
sc.Sort(ElementValue.YValue,"ASC")
```

GetInterpolatedValues() [New in v5.0]

This series method is capable of getting an estimated y value given the x value of a series. A series of elements with quantitative x axis values (meaning elements using a element.XValue or XDateTime instead of Name) can take advantage of this method.

```
[C#]
object yValue = mySeries.GetInterpolatedYValue((DateTime)xval);


[Visual Basic]
Dim yValue As Object = mySeries.GetInterpolatedYValue(DirectCast(xval, DateTime))
```

Samples:

ClickPointInterpolatedValue.aspx

ClickPointLineValue.aspx

SeriesCollection.Calculate

We will first create another series and add it to 'sc' so that we have 2 series available for our calculations.

```
[C#]
Series s2 = new Series();
s2.Elements.Add(new Element("e 1",2));
s2.Elements.Add(new Element("e 2",45));
s2.Elements.Add(new Element("e 3",23));
s2.Elements.Add(new Element("e 4",41));
s2.Elements.Add(new Element("e 5",24));
SeriesCollection sc = new SeriesCollection();
sc.Add(s);
sc.Add(s2);


[Visual Basic]
Dim s2 As New Series();
s2.Elements.Add(new Element("e 1",2))
s2.Elements.Add(new Element("e 2",45))
s2.Elements.Add(new Element("e 3",23))
s2.Elements.Add(new Element("e 4",41))
s2.Elements.Add(new Element("e 5",24))
Dim sc As New SeriesCollection();
sc.Add(s)
sc.Add(s2)
```

Now we will use the SeriesCollection.Calculate method which returns a single series.

```
 [C#]
Series s3 = sc.Calculate("Sum Series",Calculation.Sum);
sc.Add(s3);


[Visual Basic]
Dim s3 As Series = sc.Calculate("Sum Series",Calculation.Sum)
sc.Add(s3)
```

Our derived series now contains the sums of each 'element group' or 'elements with the same name'. The values of series s3 will be '7,60,46,54,58'.

Calculation Shortcuts

You can also use shortcuts to add calculated series to the chart.

```
[C#]
Chart.SeriesCollection.Add(s);
Chart.SeriesCollection.Add(s2);
Chart.SeriesCollection.Add(Calculation.Sum); // This will add a sum series to the chart.


[Visual Basic]
Chart.SeriesCollection.Add(s)
Chart.SeriesCollection.Add(s2)
Chart.SeriesCollection.Add(Calculation.Sum) ' This will add a sum series to the chart.
```

Series and element operators (+ - * /)

Series and elements can be manipulated with operators when using c#. For example, the previous code snippet can also be achieved in this way:

```
[C#]

Chart.SeriesCollection.Add(s);
Chart.SeriesCollection.Add(s2);
Chart.SeriesCollection.Add(s1+s2); // This will add a sum series to the chart.


[Visual Basic]
```

```
' Visual Basic operators will be available with .net framework 2.0, however
' methods are available for this functionality.
Chart.SeriesCollection.Add(s)
Chart.SeriesCollection.Add(s2)
Chart.SeriesCollection.Add(s1.Add(s2)) // This will add a sum series to the chart.
```

The operators can also be used with numbers. Let's say you have a chart that shows the GDP of countries. You can do the following: mySeries *= .000000001; If the value of an element in this series was 7,000,000,000 (seven billion), after this operation the value will be 7. On the axis label we can then say 'GDP (Billions)'. Operators are also useful in situations that require complex manipulation of data such as rates.

Manually iterating elements

We can also iterate through our series collection to do custom manipulation. The following code snippet will test each element and if the value is over 35, the color of that element will become red.

```
[C#]
foreach(Series mySeries in sc)
  foreach(Element myElement in mySeries.Elements)
      if(myElement.YValue > 35)
          myElement.Color = Color.Red;


[Visual Basic]
Dim mySeries As Series
Dim myElement As Element
For Each mySeries In sc
   For Each myElement in mySeries.Elements
      If myElement.YValue > 35 Then
         myElement.Color = Color.Red
      End If
   Next myElement
Next mySeries
```

Date Based Manipulation (Advanced DateGrouping)

The series class provides two methods that allow a series of elements to be split up into multiple series based on a TimeIntervalAdvanced interval and then regrouped based on another interval and a calculation. This combination of features allows replicating the DataEngine's DateGrouping functionality a lot more.

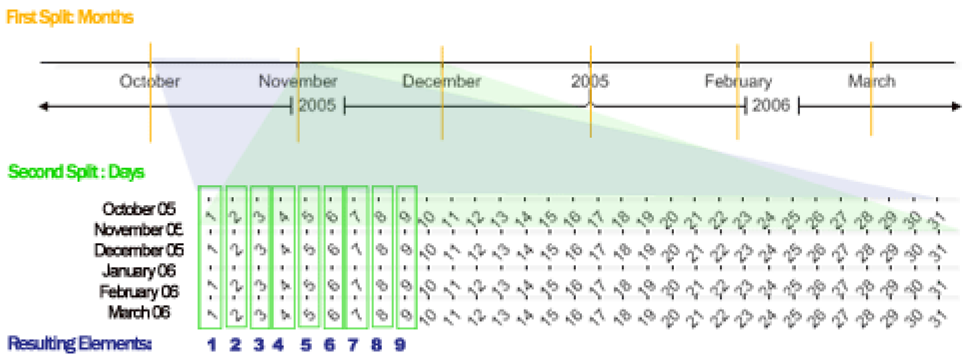
Method: SplitRegroup(TimeIntervalAdvanced firstSplit, TimeIntervalAdvanced secondSplit)
Splits the elements based on the first split, then the resulting series are grouped based on the second split. A simple analogy is a stretch of measuring tape representing a timeline. The tape is cut into sections based on (first split), then the pieces are stacked on top of each other and cut again based on the second split. The resulting stacks are evaluated down to an element based on the specified calculation.

As a real world example, consider you have a series with web site traffic represented by an element for each time the web site was visited including a date and time. This method can parse the data in a way that allows determining the traffic for each day of the week throughout the entire span of time. This means all the traffic going through the web site on any Monday would be calculated down to a single element, same for Tuesday and so on.

Method: SplitRegroupCalculate
Splits the elements based on the first split, then the resulting series are grouped based on the second split. A simple analogy is a stretch of measuring tape representing a timeline. The tape is cut into sections based on (first split), then the pieces are stacked on top of each other and cut again based on the second split. The resulting stacks are evaluated down to an element based on the specified calculation.

As a real world example, consider you have a series with web site traffic represented by an element for each time the web site was visited including a date and time. This method can parse the data in a way that allows determining the traffic for each day of the week throughout the entire span of time. This means all the traffic going through the web site on any Monday would be calculated down to a single element, same for Tuesday and so on.

This illustration shows how the data is arranged.



7.1 Element Sorting

Introduction

This tutorial will describe how elements are sorted by default and how their sort order can be manipulated. If the elements have x axis values such as numbers or dates, sorting is not very useful because the sequential axis will determine their position, however, when elements use string names for the x axis sorting is much more useful.

Quantitative Axis Sorting

Elements with numeric or dateTime x values and a numeric or dateTime x axis will automatically be sorted sequentially and fall into place where the values appear on the axis. Elements may contain names as well as x values such as dates, however, if both names and values exist, the x axis will default to a category axis which shows names of the elements. To force the axis to use a time scale the following code can be used.

```
[C#]
Chart.XAxis.Scale = Scale.Time;


[Visual Basic]
Chart.XAxis.Scale = Scale.Time
```

If the elements have names and numbers using Scale.Normal will have the same effect causing the x axis to become a numeric axis.
For more information on element values and axis scales see the **Elements and Axis Scales (on-line documentation)** tutorial.

The chart mentor can be helpful in this case as it can detect your data types, axis scales and report back suggestions.

Category Axis Sorting

Single Series Default Sorting

How They Behave
A single series of elements with names instead of x values will be drawn on the chart in the order the elements are added to the series.

TIP:

- To reverse the order of elements on an axis, it can be done quickly by using Chart.XAxis.InvertScale = true

Multiple Series Default Sorting

If multiple series are added to the chart with the same element names and in the same order the chart will draw them in the order they are added. When multiple series with elements that are different are added, the chart performs some smart grouping functionality that processes the sort order.This section will describe the element name grouping functionality.

As an example case, elements will be provided for the chart in a segmented order.

3 Series, each with 4 elements:

Series 1:			C	D	E	F		

Series 2:					E	F	G	H
Series 3:	A	B	C	D				

SeriesCollection.SmartGrouping = true (Default)

Intended order:

A	B	C	D	E	F	G	H

.netCHARTING Will analyze this list and determine what the true intended order should be. This does not employ sorting, it looks for patterns in element orders of different series to determine the original order. In this case the available element names offer enough information to determine the sort order to be the above.

SeriesCollection.SmartGrouping = false

When false, the element order will be based on the orders in which series elements are provided. The above example will result in this order:

C D E F G H A B

The first series defines the order to be CDEF then the second series has two new element names which are added at the end GH. The final series offers AB as new names and that will be added at the end and so on.

⚠ Despite any sorting performed on a series, if the x axis is a category axis, the above processing will be applied.

Custom Sorting

Series.Sort (Sort Elements)

Elements within a series can be sorted by any value in ascending or descending order.

```
[C#]
// Sort the series by name
s.Sort(ElementValue.Name, "DESC");
// Sort the series by YValue
s.Sort(ElementValue.YValue, "DESC");


[Visual Basic]
' Sort the series by name
s.Sort(ElementValue.Name, "DESC")
' Sort the series by YValue
s.Sort(ElementValue.YValue, "DESC")
```

SeriesCollection.Sort (Sort Series)

Series within a SeriesCollection can also be sorted. This does not affect the element sort order, only the order of the series within the collection. The sorting is based on the sum of any element values within a series.

```
[C#]
mySeriesCollection.Sort(ElementValue.YValue,"DESC");


[Visual Basic]
mySeriesCollection.Sort(ElementValue.YValue,"DESC")
```

SeriesCollection.SortElementGroups (Sort Element Groups)

Element groups are elements belonging to different series but sharing a name. It is the same as the requirement for a stack chart's stacks. Only element groups can be stacked. In effect, this can be considered a way to sort the columns of a stacked chart.

```
[C#]
mySeriesCollection.SortElementGroups(ElementValue.YValue,"ASC");


[Visual Basic]
mySeriesCollection.SortElementGroups(ElementValue.YValue,"ASC")
```

Sample: SortElementStacks.aspx

8 Analysis Engine

Introduction

The .netCHARTING analysis engine provides many statistical calculations and financial indicators allowing quick and accurate analysis of your data.

Getting Data

The first step is to get a series of data to analyze. Please see the following tutorials as a reference on data acquisition.

- **Data Tutorials (on-line documentation)**
- **Financial Charts (on-line documentation)**

The features discussed in this section require that series objects are populated with elements. This does not happen automatically if the DataEngine is not used. Please see **this kb (http://dotnetcharting.com/kb/article.aspx?id=10392)** for more information.

Calculated series and financial indicators often use multiple chart areas. For more info, see:

- **Multiple Chart Areas (on-line documentation)**

Analysis Engine

Three classes provide the methods used for analysis.

- **FinancialEngine**
- **StatisticalEngine**
- **ForecastEngine**

The following code snippet demonstrates calculating the mean of a series collection.

```
[C#]
...
SeriesCollection mySeries = ...; // Original seriesCollection
Series resultSeries = StatisticalEngine.Mean(mySeries);

Chart.SeriesCollection.Add(resultSeries);
```

```
[Visual Basic]
...
Dim mySeries As SeriesCollection = ...(seriesCollection)... ' Original seriesCollection
Series resultSeries = StatisticalEngine.Mean(mySeries)
Chart.SeriesCollection.Add(resultSeries)
```

Many samples are available under the Features / Analysis Engine section that demonstrate this functionality.

ForecastEngine

The forecast engine contains a set of simple methods to perform common forecasting operations. However, the class also contains an advanced set of methods for users familiar with the mathematical formulas.

IndicatorOptions

Both analysis classes contain a static **Options** class that provides additional options for use with the analysis engine.

- **PopulateSubValues**
  When a series is calculated down to a single element, the element values of the original series can be included in the resulting element as **SubValues ('ErrorBars and other SubValues' in the on-line**

documentation).

> A similar feature is also available in the dataEngine: **PopulateDateGroupingSubValues Property**

- **MatchColors**
  Gets or sets a value that indicates whether calculated elements will use the same colors that are assigned to original series. Both, the original and derived data objects must be placed on the same chart to enable this feature because colors are generally based on a palette at runtime.

The following sample demonstrates using the statistical engine's indicator options.

```
[C#]
...
SeriesCollection mySeries = ...; // Original seriesCollection

StatisticalEngine.Options.PopulateSubValues = true;
Element resultElement = StatisticalEngine.Mean(mySeries);

Chart.SeriesCollection.Add(resultElement);


[Visual Basic]
...
Dim mySeries As SeriesCollection  ... // Original series

StatisticalEngine.IndicatorOptions.PopulateSubValues = True
Element resultElement = StatisticalEngine.Mean(mySeries)

Chart.SeriesCollection.Add(resultSeries)
```