

# DashboardLayout for WinForms

2022.04.21 更新

グレースィティ株式会社

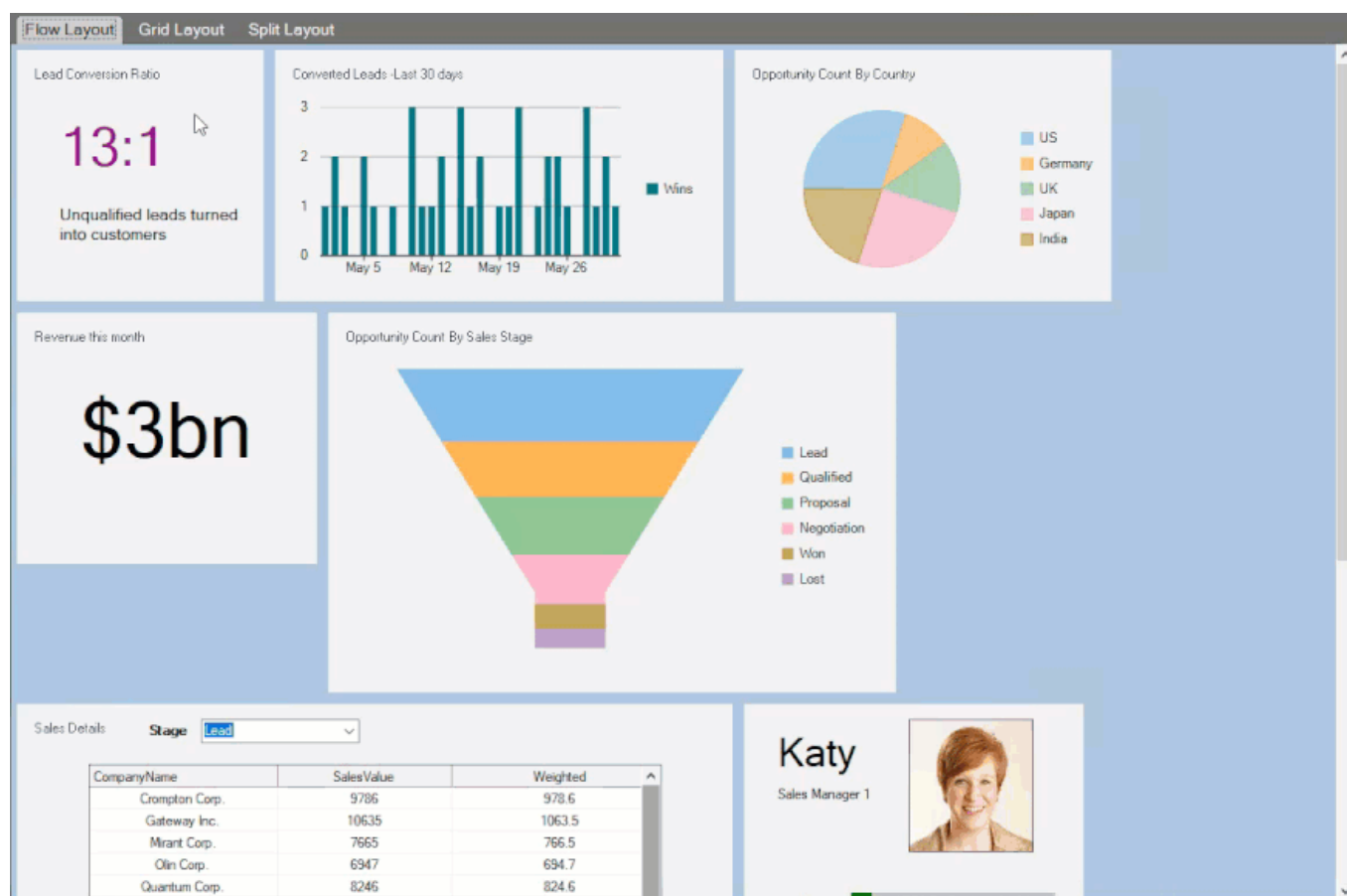
## 目次


<a href="#">DashboardLayout for WinForms</a>	2
<a href="#">主な特長</a>	3
<a href="#">基本クラス</a>	4
<a href="#">要素</a>	5-6
<a href="#">クイックスタート</a>	7-8
<a href="#">DashboardLayoutの操作</a>	9
<a href="#">レイアウト</a>	9
<a href="#">子コンテナのヘッダー</a>	9-10
<a href="#">ドラッグアンドドロップ</a>	10-12
<a href="#">ツールアイコン</a>	12-13
<a href="#">サイズ変更</a>	13-15
<a href="#">XMLシリアル化</a>	15-16
<a href="#">ダッシュボードのスタイル</a>	17-18
<a href="#">チュートリアル</a>	19
<a href="#">フローレイアウトでダッシュボードの作成</a>	19-22
<a href="#">グリッドレイアウトでダッシュボードの作成</a>	22-25
<a href="#">分割レイアウトでダッシュボードの作成</a>	25-28
<a href="#">カスタムコンテキストメニューの作成</a>	28-32
<a href="#">子コンテナのヘッダのカスタマイズ</a>	32-37

## DashboardLayout for WinForms

**DashboardLayout for WinForms** is a layout control that allows you to create dynamic dashboards for interactive data visualization. It allows you to organize and present data in a consolidated form with the help of images, grids, charts, maps, etc in different layouts. This makes it easy for you to monitor the presented information.

The DashboardLayout control acts as a container which lets you dynamically place controls within tiles also called child containers. These child containers can be arranged in different layouts in the layout control contained within the dashboard control. The control lets you resize and rearrange these child containers at runtime to create an ideal workspace.



Documentation	Blogs
<a href="#">クイックスタート</a> <a href="#">DashboardLayoutの操作</a> <a href="#">ダッシュボードのスタイル</a> <a href="#">チュートリアル</a>	<a href="#">Dashboard Layout for WinForms</a> <a href="#">Create BI Dashboard using DashboardLayout for WinForms</a> <a href="#">Integrate Google Analytics Data in DashboardLayout</a>
Videos	Demo Samples
<a href="#">ComponentOne Videos</a>	<a href="#">WinForms DashboardLayout</a>
 Note: ComponentOne DashboardLayout for WinForms is compatible with both .NET and .NET framework.	
API References	
<b>DashboardLayout .NET Framework Assembly</b>	<b>DashboardLayout.NET Assembly</b>

## 主な特長

DashboardLayout provides many different features that enable the developers to build intuitive and professional-looking dashboards. The main features for DashboardLayout are as follows:

- **Use different layouts**

DashboardLayout provides three types of layouts, namely Flow, Grid and Split. These layouts specify the arrangement of the child containers in different ways on the DashboardLayout control.

- **Customize child container headers**

DashboardLayout allows customization in child container's header by either providing caption or adding custom objects such as buttons and icons, to it.

- **Drag and drop child containers**

DashboardLayout lets you drag a child container at runtime. When you hover the mouse over an empty space of the child container, a hand tool appears on its surface to drag it. DashboardLayout also displays a visual cue for possible layout which helps users to appropriately place the selected child container at the desired location.

- **Resize child containers**

DashboardLayout supports resizing of child containers with the help of mouse at both, design-time and runtime. The child containers can also be maximized or restored at runtime for better viewing of the contents.

- **Perform multiple functions with ToolIcon**

DashboardLayout provides a multi functional tool called Tool-icon, which appears as an icon in the header or on the surface of the child container on hovering the mouse over it. It can be used to drag and drop the child containers. Moreover, tool-icon provides a context menu with options to maximize or hide a child container.

- **Serialization/Deserialization**

DashboardLayout offers XML serialization by providing the functionality to save the layout properties of the control to an XML file or stream and load them from another XML file or stream.

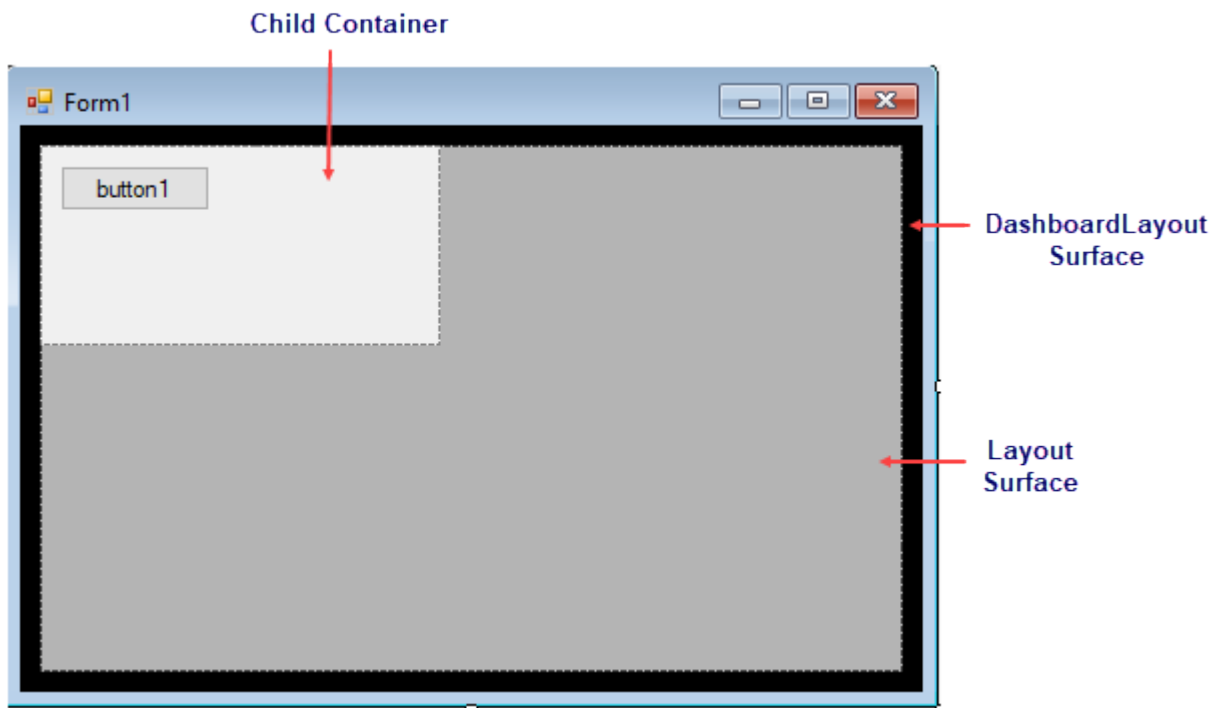
## 基本クラス

The following table lists some of the key classes and their properties. You can click on the cross-references to get to the API documentation that provides detailed description for the key members.

<b>C1DashboardLayout</b>
<b>Properties:</b> Items, LayoutType, Options, Styles <b>Methods:</b> LoadLayout, SaveLayout
<b>DashboardItem</b>
<b>Property:</b> Caption
<b>DashboardOptions</b>
<b>Property:</b> ContextMenuStrip

## 要素

The DashboardLayout control consists of the following elements as shown in the image.



## DashboardLayout

DashboardLayout is represented by **C1DashboardLayout** class. It serves as the main container with an attached layout that defines how the child containers are arranged.

## Layout

The DashboardLayout element uses layout controls to create a layout for arranging child containers containing controls on the surface of the DashboardLayout control. It supports flow, grid and split layout types and each of them behave differently on resizing or arranging the child containers. Dashboard allows you to specify the type of layout using **LayoutType** property of the **C1DashboardLayout** class. The control communicates with the layout controls via the **ILayout** interface. This interface can also be implemented for creating custom layouts.

## Child containers

When a control is dragged on the DashboardLayout control, a child container is automatically created to hold that control. These child containers can contain arbitrary content like text, images or any UI controls like charts, grids, maps etc. Depending on the layout you specify using the **LayoutType** property, the child container is either Panel class or C1SplitterPanel class. Thus, if the LayoutType property is set to **LayoutType.Grid** or **LayoutType.Flow**, the child container is represented by the Panel class and if it is set to **LayoutType.Split**, the child container is represented by the **C1SplitterPanel** class.

The child containers can be accessed using Items property of the C1DashboardLayout class as depicted below:

### Visual Basic

```
((C1SplitterPanel)
(c1DashboardLayout1.Items["Container1"].ItemContainer)).Dock =
PanelDockStyle.Left
((C1SplitterPanel)
```

```
(c1DashboardLayout1.Items["Container1"].ItemContainer)).Text = "Splitter  
Panel"  
(C1SplitterPanel)  
(c1DashboardLayout1.Items["Container1"].ItemContainer)).Collapsible =  
true  
(C1SplitterPanel)  
(c1DashboardLayout1.Items["Container1"].ItemContainer)).HeaderBackColor  
= Color.Green
```

C#

```
((C1SplitterPanel)  
(c1DashboardLayout1.Items["Container1"].ItemContainer)).Dock =  
PanelDockStyle.Left;  
(C1SplitterPanel)  
(c1DashboardLayout1.Items["Container1"].ItemContainer)).Text = "Splitter  
Panel";  
(C1SplitterPanel)  
(c1DashboardLayout1.Items["Container1"].ItemContainer)).Collapsible =  
true;  
(C1SplitterPanel)  
(c1DashboardLayout1.Items["Container1"].ItemContainer)).HeaderBackColor  
= Color.Green;
```

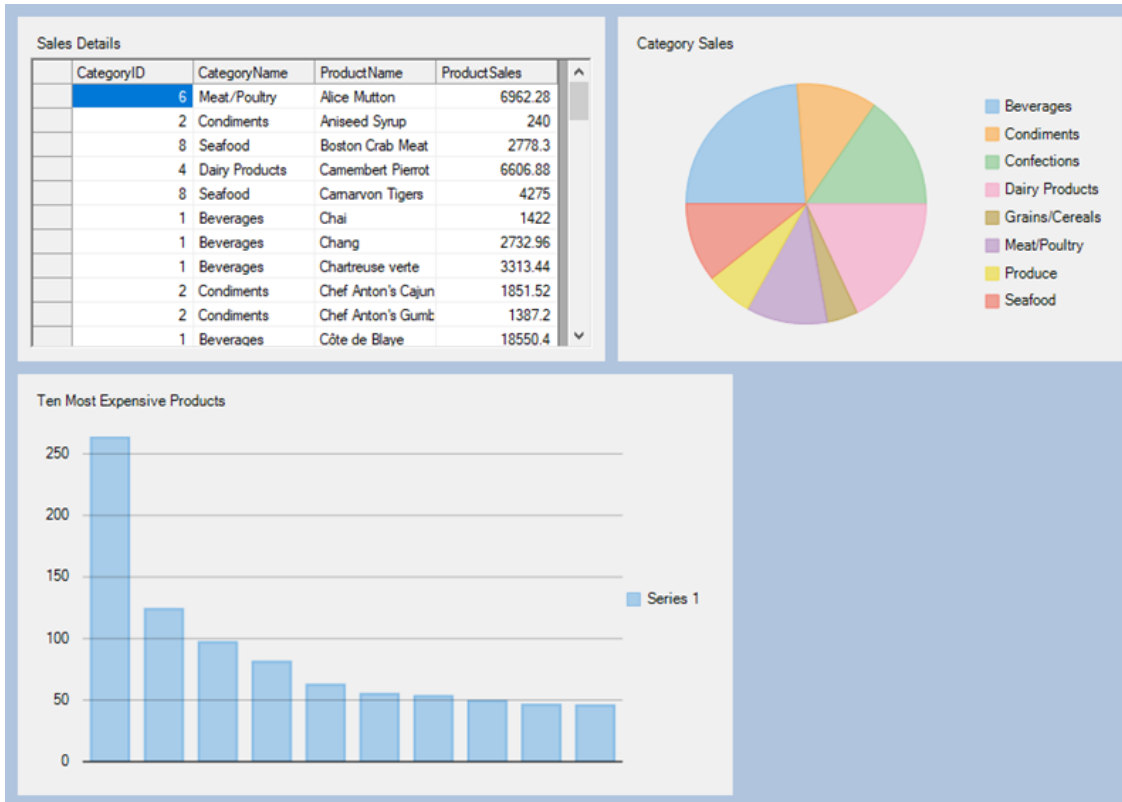
# DashBoardLayout for WinForms

## クイックスタート

This quick start will guide you through the steps of adding C1DashboardLayout control to create a simple dashboard application. Follow the steps below to get started:

1. **Setting up the application**
2. **Create a datasource for DashboardLayout**
3. **Add controls to DashboardLayout**

The following image shows a dashboard created using the C1DashboardLayout control.



### Step 1: Setting up the application

1. Create a new WinForms application.
2. Add the C1DashboardLayout control to the form.
3. In the **Properties** window, navigate to **LayoutType** property and set it to **Flow**.  
Observe: The C1DashboardLayout control is docked in the form and a layout of the type Flow is attached to it.
4. Set the height of the form to 1000 and width to 1100.

### Back to Top

### Step 2: Create a datasource for DashboardLayout

1. Go to the **Project** menu and select **Add New Data Source** from the Project dropdown menu.  
The Data Source Configuration Wizard dialog box appears.
2. Select **Database** and click **Next**.
3. Select **Dataset** and click **Next**.
4. Click **New Connection**.
5. Select **Data source** as **Microsoft Access Database File (OLEDB)**.
6. Click **Browse** to add a database file name as C1NWind.mdb by navigating to its default location, C:\Users\Documents\ComponentOne Samples\Common.
7. Select C1NWind database, click **Open**, and then click **OK**.
8. Click the **Next** button to continue. A dialog box appears asking if you would like to add the data file to your project and modify the connection string. Since it is not necessary to copy the database to your project, click **No**.
9. Verify that **Yes, save the connection as** check box is checked and click **Next** to continue. The connection string is saved as C1NwindConnectionString.

10. Expand the **Views** node and select Category Sales for 2014, Sales by Category and Ten Most Expensive Products objects.
11. Click **Finish**.  
Observe: C1NwindDataSet.xsd is added to your project.

**Back to Top**

## Step 3: Add controls to DashboardLayout

1. Drag and drop the C1FlexGrid control on the DashboardLayout.  
Observe: A child container (Panel) automatically gets created under FlexGrid. It positions itself at the upper left corner of the flow layout since its **FlowDirection** property is set to **LeftToRight** by default.
2. Navigate to the **DataSource** property of the FlexGrid control and select C1NwindDataSet to bind the control to the data source.  
Observe: A binding source gets created and the **DataSource** property is set to c1NwindDataSetBindingSource.
3. Set the **DataMember** property to **Sales by Category**.
4. Now, drag and drop a FlexPie on the DashboardLayout.  
Observe: Since the width of the C1DashboardLayout has not yet exhausted, the child container containing the FlexPie positions itself next to the FlexGrid maintaining the default LeftToRight flow direction.
5. Navigate to the **DataSource** property of FlexPie and select the already created c1NwindDataSetBindingSource.
6. Set the **DataMember** property to **Category Sales for 2014**, **Binding** property to **CategorySales** and **BindingName** property to **CategoryName**.
7. Drag and drop the FlexChart control on the DashboardLayout.  
Observe: The child container containing the FlexChart control is wrapped to the next row as the width of the DashboardLayout has exhausted and the **WrapContents** property of the flow layout is set to true. Otherwise, the child container gets clipped.
8. Navigate to the **DataSource** property of FlexChart and select c1NwindDataSetBindingSource. Set the **DataMember** property to **Ten Most Expensive Products**, **Binding** property to **UnitPrice** and **BindingX** property to **TenMostExpensiveProducts**.
9. Add the following code in **Form1\_Load** event to populate the controls with data:

Visual Basic

```
Private Sub Form1_Load(ByVal sender As Object, ByVal e As EventArgs)
    Dim salesByCategoryAdapter As
C1NwindDataSetTableAdapters.Sales_by_CategoryTableAdapter = New
C1NwindDataSetTableAdapters.Sales_by_CategoryTableAdapter()
    salesByCategoryAdapter.Fill(c1NwindDataSet.Sales_by_Category)
    Dim categorySalesAdapter As
C1NwindDataSetTableAdapters.Category_Sales_for_2014TableAdapter = New
C1NwindDataSetTableAdapters.Category_Sales_for_2014TableAdapter()
    categorySalesAdapter.Fill(c1NwindDataSet.Category_Sales_for_2014)
    Dim expensiveProductsAdapter As
C1NwindDataSetTableAdapters.Ten_Most_Expensive_ProductsTableAdapter = New
C1NwindDataSetTableAdapters.Ten_Most_Expensive_ProductsTableAdapter()
    expensiveProductsAdapter.Fill(c1NwindDataSet.Ten_Most_Expensive_Products)
End Sub
```

o C#

```
private void Form1_Load(object sender, EventArgs e)
{
    C1NwindDataSetTableAdapters.Sales_by_CategoryTableAdapter salesByCategoryAdapter =
        new C1NwindDataSetTableAdapters.Sales_by_CategoryTableAdapter();
    salesByCategoryAdapter.Fill(c1NwindDataSet.Sales_by_Category);

    C1NwindDataSetTableAdapters.Category_Sales_for_2014TableAdapter categorySalesAdapter =
        new C1NwindDataSetTableAdapters.Category_Sales_for_2014TableAdapter();
    categorySalesAdapter.Fill(c1NwindDataSet.Category_Sales_for_2014);

    C1NwindDataSetTableAdapters.Ten_Most_Expensive_ProductsTableAdapter expensiveProductsAdapter =
        new C1NwindDataSetTableAdapters.Ten_Most_Expensive_ProductsTableAdapter();
    expensiveProductsAdapter.Fill(c1NwindDataSet.Ten_Most_Expensive_Products);
}
```

10. Run the application. A simple dashboard with three child containers is displayed.

**Back to Top**

## DashboardLayoutの操作

This section comprises all the features and auxiliary functionality offered by the DashboardLayout control.

### レイアウト

Learn about different layouts supported by DashboardLayout.

### 子コンテナのヘッダー

Learn about the child container header.

### 子コンテナのコンテキストメニュー

Learn how to display the child container context menu and customize it.

### ドラッグアンドドロップ

Learn how drag and drop works in DashboardLayout.

### ツールアイコン

Learn about the tool-icon and how to set its location.

### サイズ変更

Learn about the resizing behavior of child containers in different layouts.

### ダッシュボードのスタイル

Learn how to style the container, child container, and tool-icon.

### XMLシリアル化

Learn how DashboardLayout supports serialization.

## レイアウト

DashboardLayout supports the following three types of layouts which determine the way child containers are arranged:

- フローレイアウト
- グリッドレイアウト
- 分割レイアウト

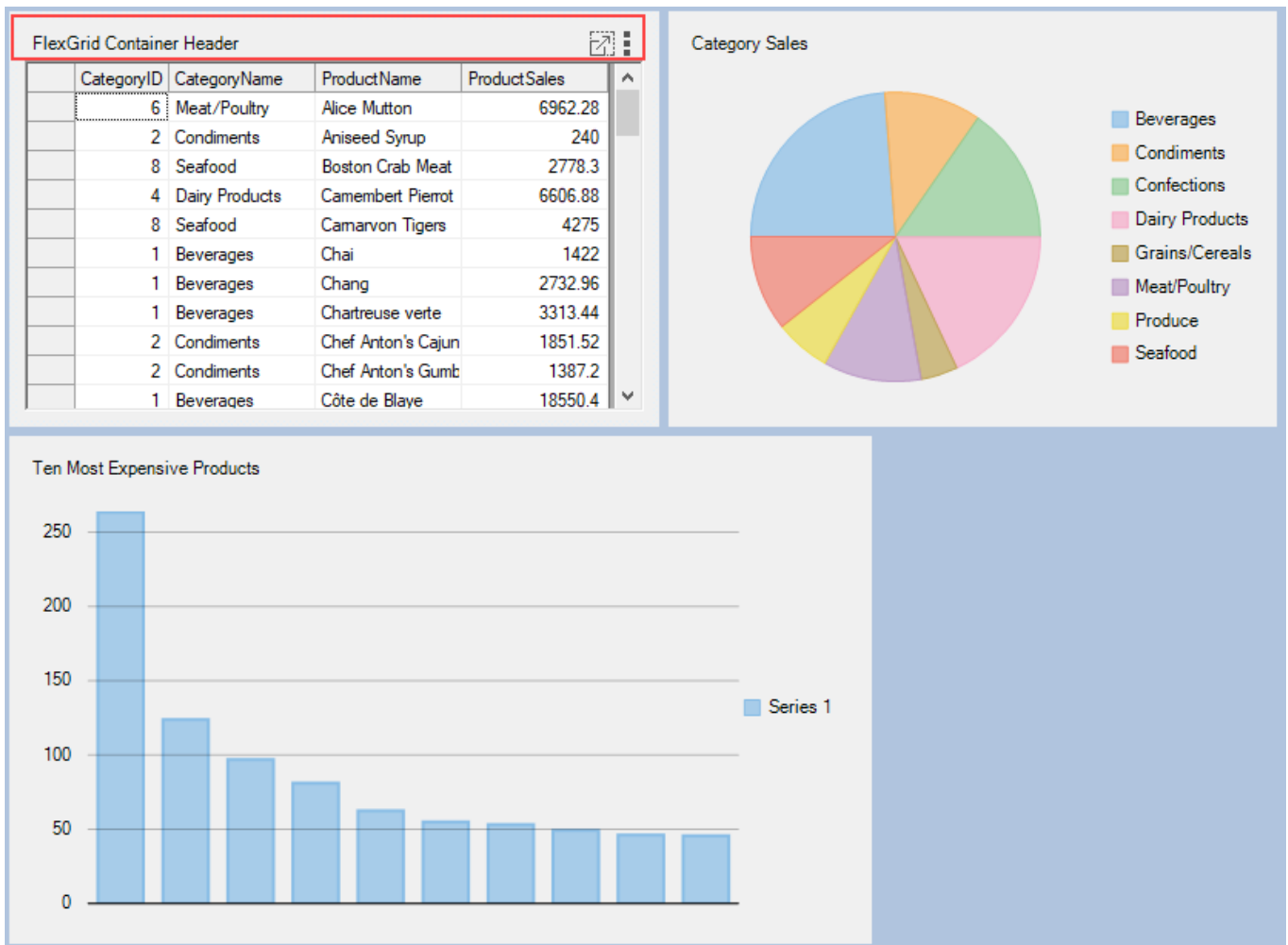
DashboardLayout allows you to set these layout types using its **LayoutType** property. This property accepts the values from **LayoutType** enumeration which specifies following options. When you select the layout type, it gets attached to the DashboardLayout control.

- Flow
- Grid
- Split
- Custom

## 子コンテナのヘッダー

DashboardLayout allows you to display the child container header by using **SetCaption** method of the **C1DashboardLayout** class. This method sets caption for the header so as to make the header visible. The header consists of caption text, a maximize icon and tool-icon with three dots positioned vertically, which can be used to hide or drag the child container.

The following image shows a header in the child container.



For setting header on a child container, use the following code:

C#

```
c1DashboardLayout1.SetCaption(c1DashboardLayout1.Items[0].ItemContainer,
"FlexGrid Container Header");
```

You can also add custom elements to the child container headers using `Options.HeaderElements` at design time.

For styling child container headers, refer [ダッシュボードのスタイル](#) topic.

## ドラッグアンドドロップ

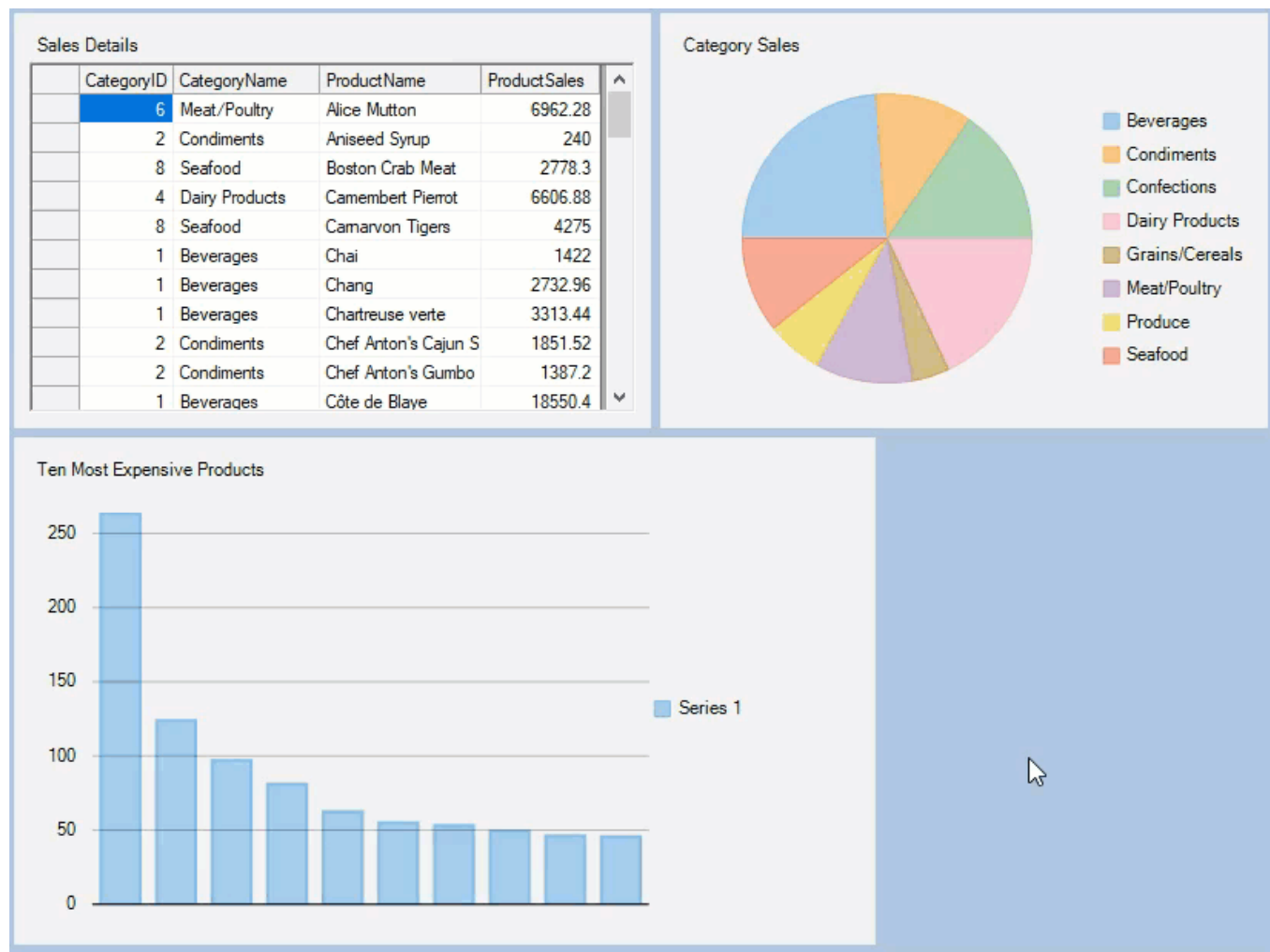
DashboardLayout supports drag drop operation of the child containers. On performing drag drop operation, the manner in which child containers reposition themselves differ for every layout. In flow layout, when a child container is dragged, all the remaining child containers reposition themselves in the flow direction set for the layout. On the other hand, when you drag a child container in the grid layout, the dragged child container replaces its position with another child container in its cell. And, when you drag a child container in split layout, it shifts to the other panel in the mouse navigation path and all the other child containers change their positions as well.

You can drag and drop a child container using two methods, using hand tool and tool-icon. The hand tool appears when you hover the mouse over an empty space in the child container. This tool lets you drag and drop a child container to another position on the dashboard. The hand tool can not be used when child container is docked into the control as there is no empty space. As an alternative, you can use the tool-icon to perform the drag drop

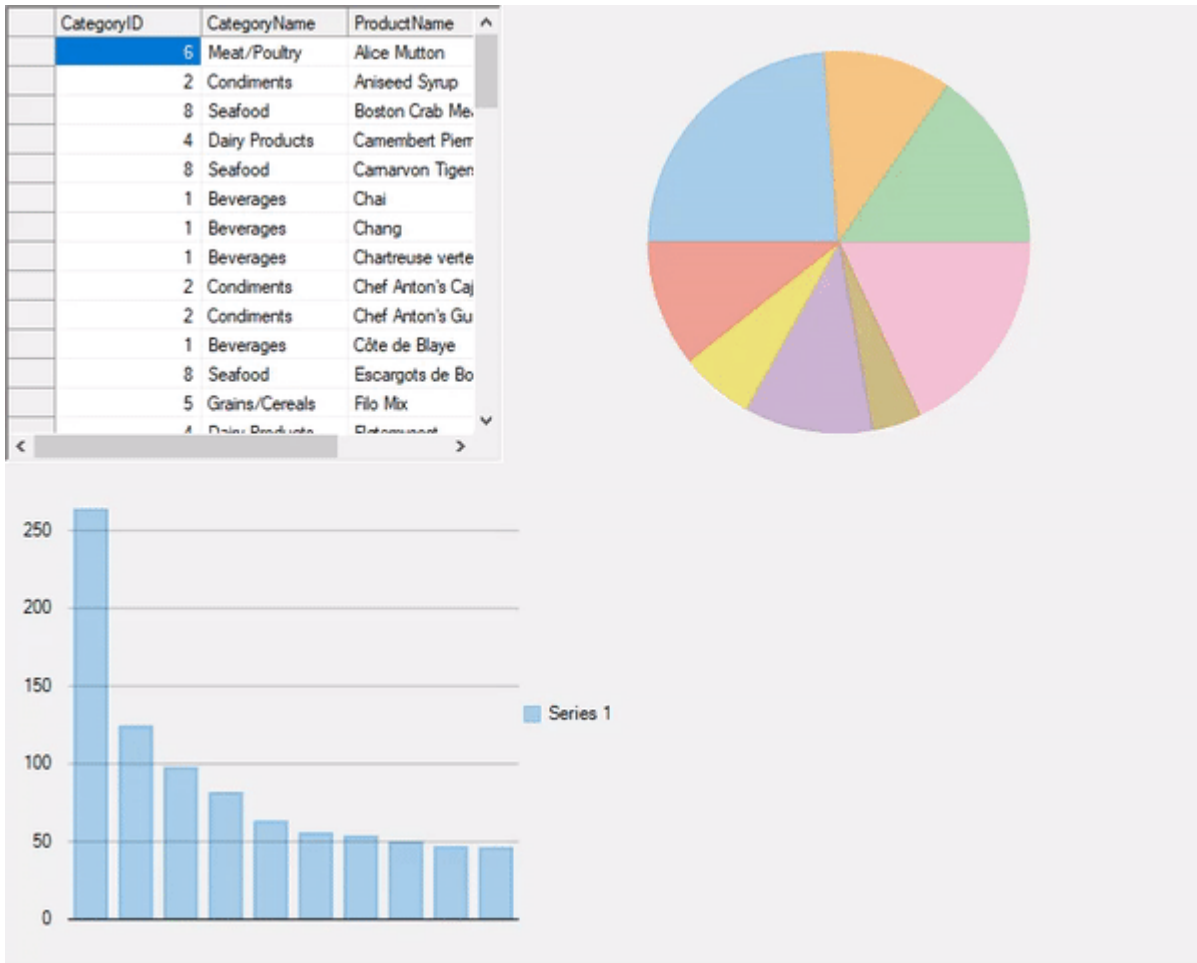
# DashBoardLayout for WinForms

operation by holding the cursor on the tool-icon.

The following GIF shows drag and drop operation performed using hand tool:



The following GIF shows drag and drop operation performed using tool-icon:



## ツールアイコン

Tool-icon is a multifunctional tool which appears as an icon either on the surface or in the header of the child container when it is selected with mouse hover. It appears as three dots positioned vertically.

The tool-icon can be used to drag and drop a child container by hovering the mouse over it and dragging the child container to another position. On clicking, the tool-icon opens a context menu which by default, displays the options for maximizing or hiding a child container. You can also double-click the tool-icon to maximize or restore the child container.

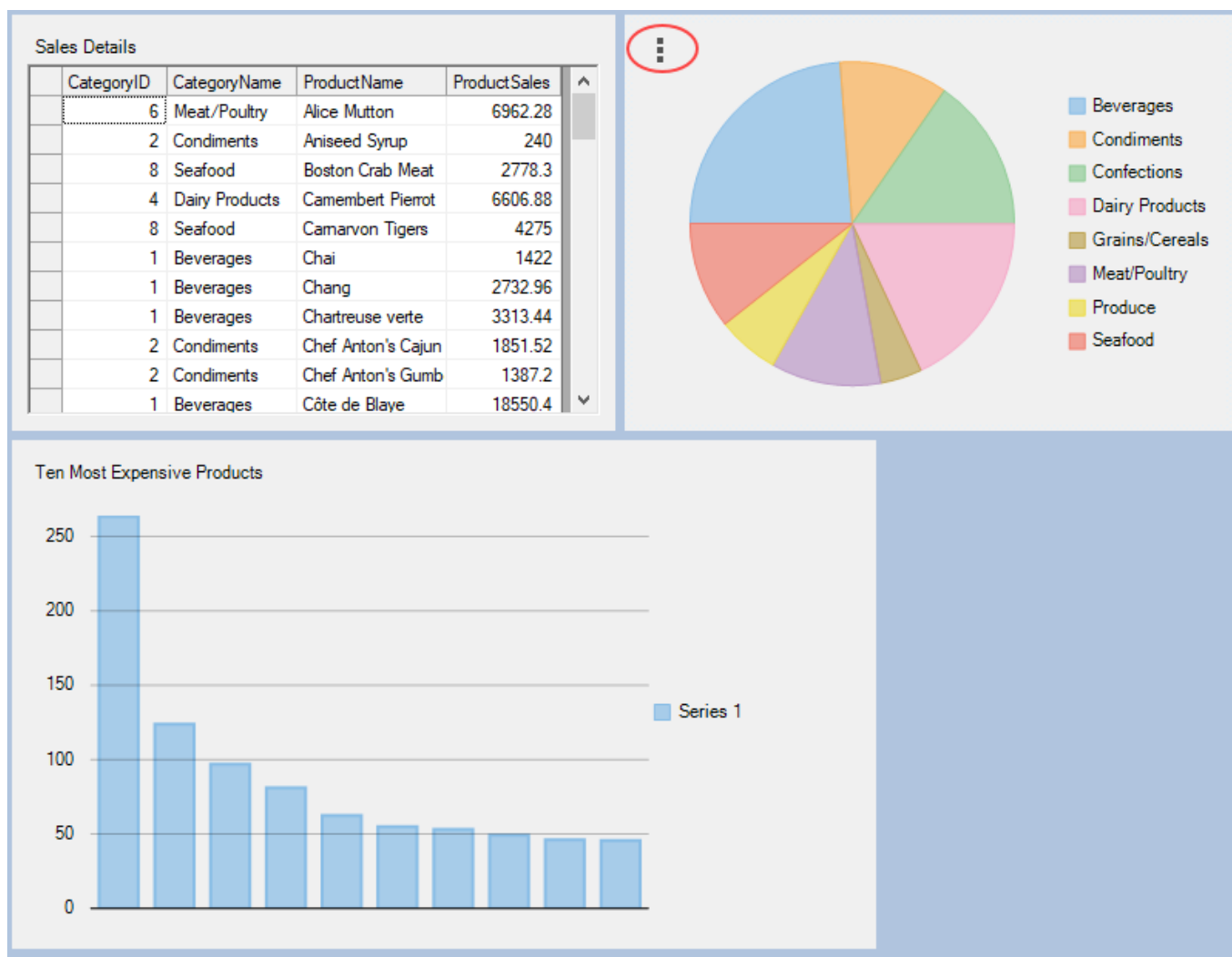
Furthermore, DashboardLayout allows you to set the location of the tool-icon on the child container using **ToolIconAppearance** property of the **DashboardOptions** class. This property accepts one of the following values from the **ToolIconAppearance** enumeration:

- None
- UpperLeft
- UpperRight
- LowerLeft
- LowerRight

By default, the value of **ToolIconAppearance** property is set to **None** so the tool-icon is not visible.

The following image shows the tool-icon located at the upperleft corner of the child container:

# DashboardLayout for WinForms



The following code shows an example to set the location of the tool-icon:

C#

```
c1DashboardLayout1.Options.ToolIconAppearance =  
C1.Win.Layout.ToolIconAppearance.UpperLeft;
```

When the header is set on a child container, the tool-icon appears on the right side of the header. Otherwise, the tool-icon appears on the surface of the child container. If the tool-icon is not visible, the context menu appears on right clicking the child container.

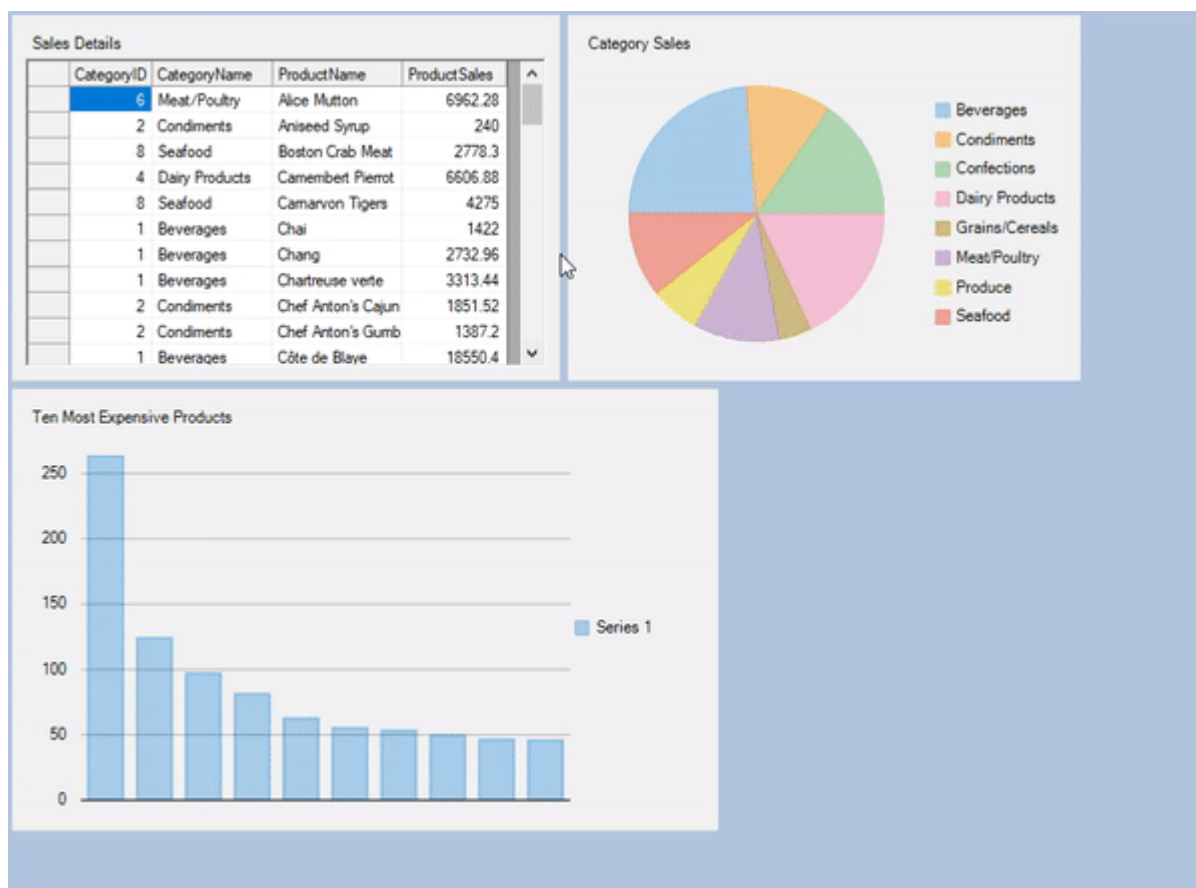
For styling the tool-icon, refer [ダッシュボードのスタイル](#) topic.

## サイズ変更

DashboardLayout allows you to resize the child containers at runtime. The resizing behavior of the child containers differ for different layouts. The behaviors are discussed below:

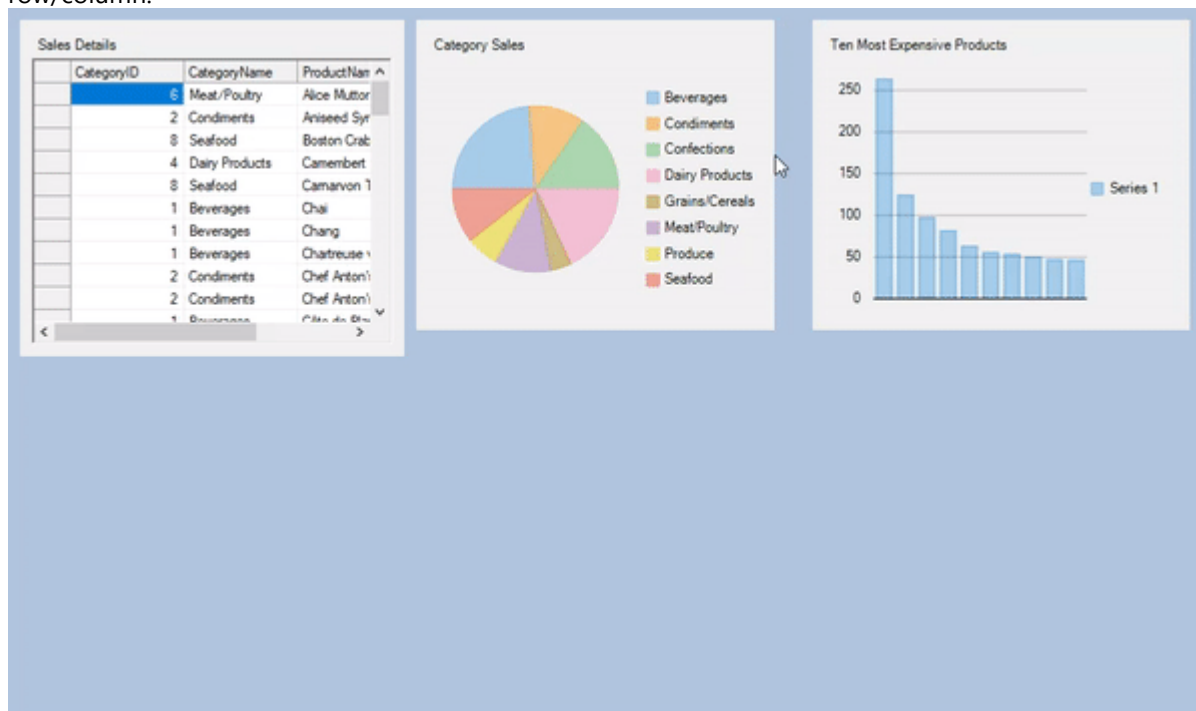
### Resizing in Flow layout

On resizing a specific child container, other child containers rearrange themselves maintaining the flow direction.



## Resizing in Grid layout

On resizing width of a specific child container, it spans the adjacent column and the other child containers rearrange themselves accordingly in the next column/row. However, on resizing height of a specific child container, it spans the row below and the other child containers rearrange themselves accordingly in the next row/column.



## Resizing in Split layout

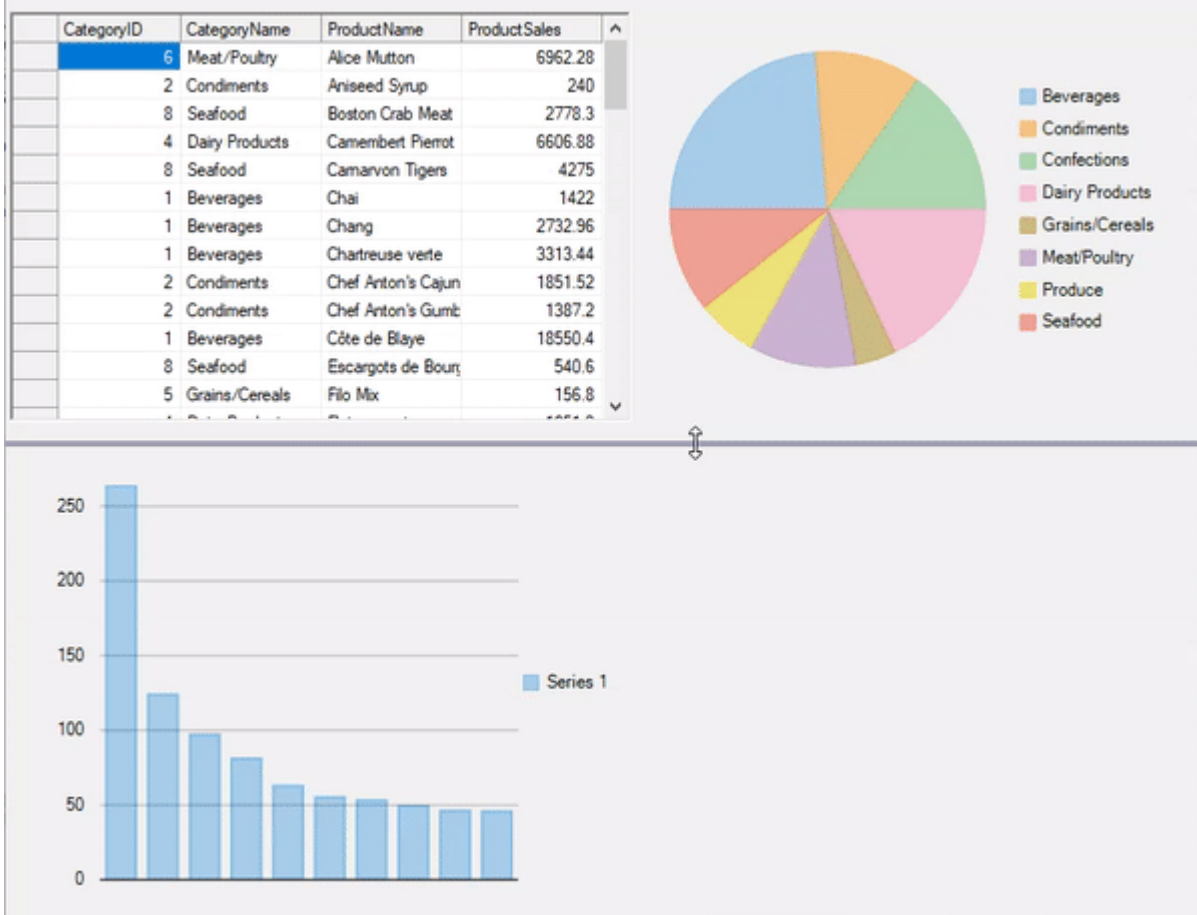
On hovering mouse over the **SplitterBar**, a resizing handle appears over the bar which can be used to resize the

# DashBoardLayout for WinForms

child containers (**C1SplitterPanel**) at runtime. However, you can specify minimum width or height of each panel to prevent the user from resizing it too small through the **MinWidth** and **MinHeight** properties of the **C1SplitterPanel** class. The following code below shows an example of setting the minimum width for the child containers:

C#

```
((C1SplitterPanel)
(c1DashboardLayout1.Items["Container1"].ItemContainer)).MinWidth =
200;
```



You can also specify the minimum and maximum size for all the child containers at once using **MinimumItemContainerSize** and **MaximumItemContainerSize** properties of the **DashboardOptions** class as illustrated in the following code:

C#

```
c1DashboardLayout1.Options.MinimumItemContainerSize = new
System.Drawing.Size(400,200);
c1DashboardLayout1.Options.MaximumItemContainerSize = new
System.Drawing.Size(600, 400);
```

## XMLシリアル化

Serialization refers to converting an object into a stream of bytes. The purpose of serialization is to save the current state of an object so that it can be recreated when required. DashboardLayout supports serialization through **SaveLayout** and **LoadLayout** methods of the **C1DashboardLayout** class. The **SaveLayout** method saves the

DashboardLayout layout properties, i.e., order and bounds of child containers, to an XML stream or file. The **LoadLayout** method loads the DashboardLayout layout properties from an XML stream or file.

The following code shows the example of saving and loading the DashboardLayout to and from an XML file:

C#

```
c1DashboardLayout1.SaveLayout("DashboardLayout.xml");  
c1DashboardLayout1.LoadLayout("DashboardLayout.xml");
```

The following code shows example of saving and loading the DashboardLayout to and from an XML stream:

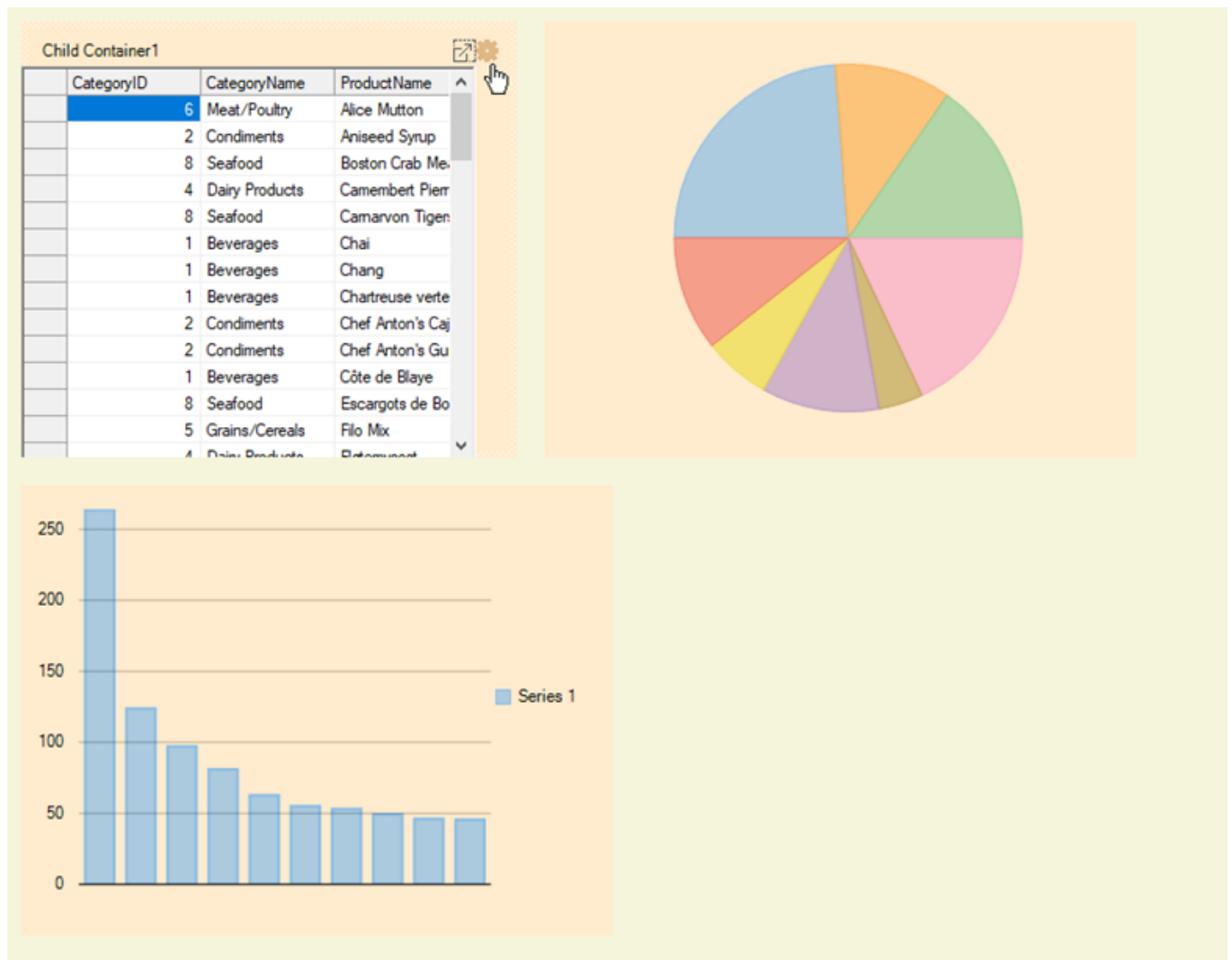
C#

```
MemoryStream stream = new MemoryStream();  
c1DashboardLayout1.SaveLayout(stream);  
stream.Seek(0, SeekOrigin.Begin);  
c1DashboardLayout1.LoadLayout(stream);
```

## ダッシュボードのスタイル

DashboardLayout allows you to customize its appearance along with the appearance of the child containers using **Styles** property. It returns an object of the class **DashboardTheme** which provides three different properties namely **Common** referring to the **CommonStyle** class and **ItemContainer** referring to the **ItemContainerStyle** class and **ContainerHeader** referring to the **ContainerHeaderStyle** class. These classes handle the styling of the DashboardLayout control and the child containers and their headers respectively.

The following image shows the styling applied to the DashboardLayout, the child containers and the tool-icon.



### Container Styling

The appearance of the DashboardLayout control can be customized using the properties exposed by the **CommonStyle** class. The following code shows an example to style the DashboardLayout:

C#

```
DashboardTheme dashboardTheme = clDashboardLayout1.Styles;  
dashboardTheme.Common.Margins = new C1.Framework.Thickness(30, 30, 30, 30);  
dashboardTheme.Common.Padding = new C1.Framework.Thickness(20, 20, 20, 20);  
dashboardTheme.Common.BackColor = Color.Beige;
```

### Item Container Styling

The appearance of the child containers or item containers can be changed using the properties exposed by the **ItemContainerStyle** class. The following code uses the properties of the **ItemContainerStyle** class to change the appearance of the child container:

C#

```
DashboardTheme dashboardTheme = c1DashboardLayout1.Styles;
dashboardTheme.ItemContainer.BackColor = Color.BlanchedAlmond;
dashboardTheme.ItemContainer.Hot.BackColor = Color.White;
dashboardTheme.ItemContainer.Margins = new
C1.Framework.Thickness(10,10,10,10);
dashboardTheme.ItemContainer.Padding = new
C1.Framework.Thickness(10,10,10,10);
```

In addition, DashboardLayout allows you to set a custom tool-icon image using **ToolIcon** property of the **ItemContainerStyle** class. The property also allows you to change the color of tool-icon image. The following code illustrates how **ToolIcon** property can be used to set a custom tool-icon and change its color:

C#

```
DashboardTheme dashboardTheme = c1DashboardLayout1.Styles;
dashboardTheme.ItemContainer.ToolIcon =
Image.FromFile(@"C:\ToolIcon.png");
dashboardTheme.ItemContainer.ToolIconColor = Color.BurlyWood;
```



**Note:** These properties can be applied to all the child containers so if you want to style a specific child container, you can use the following code snippet.

```
((Panel) (c1DashboardLayout1.Items[0].ItemContainer)).BackColor = Color.Purple;
```

## Item Container Header Styling

You can customize the appearance of the child container header using the properties exposed by the **ContainerHeaderStyle** class. These properties are applied to the headers of all the child containers. The following code illustrates the properties used for customizing the appearance of the child container:

C#

```
DashboardTheme dashboardTheme = c1DashboardLayout1.Styles;
dashboardTheme.ContainerHeader.BackColor = Color.BurlyWood;
dashboardTheme.ContainerHeader.ForeColor = Color.Brown;
dashboardTheme.ContainerHeader.Hot.ForeColor = Color.Cyan;
dashboardTheme.ContainerHeader.Hot.BackColor = Color.Red;
```

## チュートリアル

The walkthrough topics in this section are created with the assumption that you are familiar with the DashboardLayout control and know how to use it in general.

### フローレイアウトでダッシュボードの作成

Learn how to create a lead conversion dashboard using flow layout.

### グリッドレイアウトでダッシュボードの作成

Learn how to create a lead conversion dashboard using grid layout.

### 分割レイアウトでダッシュボードの作成

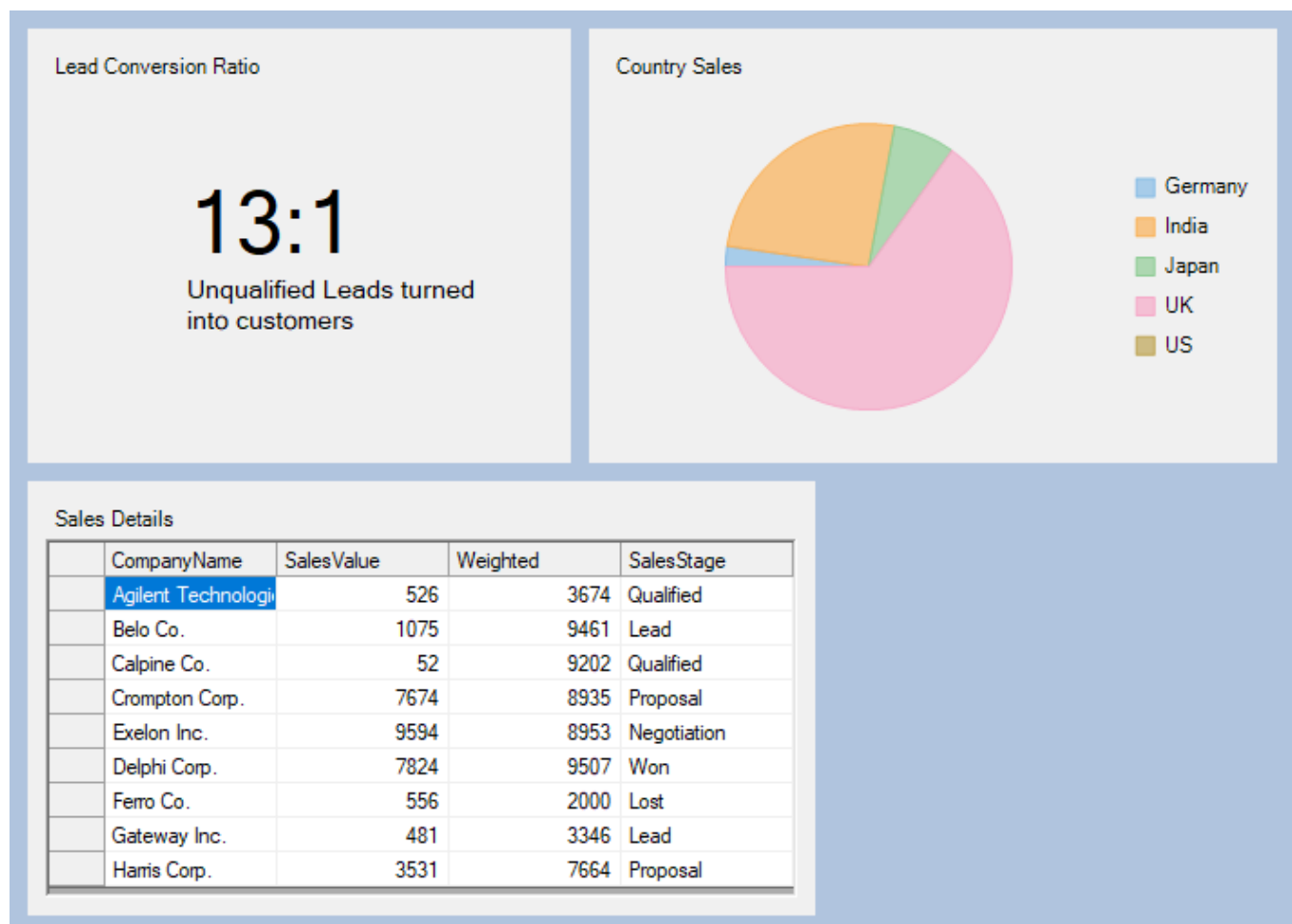
Learn how to create a lead conversion dashboard using split layout.

### カスタムコンテキストメニューの作成

Learn how to create a custom context menu.

## フローレイアウトでダッシュボードの作成

This topic guides you through the steps to create a lead conversion dashboard using flow layout. It displays the lead conversion ratio, sales in different countries in a pie chart, and sales details of companies in a FlexGrid.



1. Create a new Windows Form application.
2. In the Windows Forms Designer, set the height of the form to 600 and width to 850.
3. Drag and drop a DashboardLayout control from the Toolbox onto your form.  
Observe: By default, a layout of the type Split is attached to it.
4. Select the DashboardLayout control. In the **Properties** window, set the following properties:

Property Name	Value
---------------	-------

LayoutType	Flow
Dock	Fill

- Click inside the DashboardLayout control. The FlowContentPanel (layout control attached to the DashboardLayout) is selected.
- Drag and drop a Label control on the DashboardLayout and set the following properties:

Property Name	Value
Location	70, 70
Text	13:1
Font size	35

Observe: A child container named Panel1 of the type Panel is automatically created under the Label control and it positions itself at the upper left corner of the FlowContentPanel since its **FlowDirection** property is set to **LeftToRight** by default.

- Select **Panel1** and set the following properties:

Property Name	Value
Width	310
Height	230
Caption on c1DashboardLayout1	Lead Conversion Ratio

Now, drag and drop another Label control on Panel1 and set the following properties:

Property Name	Value
Location	75, 140
Text	Unqualified Leads turned into customers
Font size	10

- Drag and drop a FlexPie on the DashboardLayout control and set the following properties:

Property Name	Value
Dock	Fill
BindingName	CountryName
Binding	OpportunityCount

A child container named Panel3 is created under it and the child container containing the FlexPie positions itself next to Panel1.

- Select Panel3 and set the following properties:

Property Name	Value
Width	400
Height	230
Caption on c1DashboardLayout1	Country Sales

- Add a class named CountrySales.cs to the project and copy the following code to it to add data for FlexPie.

C#	copyCode
<pre>public class CountrySales</pre>	

```
{
    public CountrySales(string countryName, int opportunityCount)
    {
        CountryName = countryName;
        OpportunityCount = opportunityCount;
    }
    public CountrySales() { }
    public string CountryName { get; set; }
    public int OpportunityCount { get; set; }
    public List<CountrySales> GetData()
    {
        string[] countryNames = {"Germany", "India", "Japan", "UK" , "US" };
        List<CountrySales> countrySalesList = new List<CountrySales>();
        Random random = new Random();
        for (int i = 0; i < 5; i++)
            countrySalesList.Add(new
CountrySales(countryNames[i],random.Next(0,30)));
        return countrySalesList;
    }
}
```

11. Add the following code to Form1's Load event to populate the pie with data.

C#	copyCode
<pre>CountrySales countrySales = new CountrySales(); List&lt;CountrySales&gt; countrySalesList = countrySales.GetData(); flexPie1.DataSource = countrySalesList;</pre>	

12. Drag and drop a FlexGrid control on the DashboardLayout and set its **Dock** property to **Fill**.  
A child container named Panel5 is created under it. You can see that since the width of the C1DashboardLayout has got exhausted and the **WrapContents** property of the FlowContentPanel is set to true hence the child container containing the FlexGrid is wrapped to the next row, else it would have been clipped.
13. Select Panel5 and set the following properties:

Property Name	Value
Width	510
Height	260
Caption on c1DashboardLayout1	Sales Details

14. Add a class named SalesDetails.cs to the project and copy the following code to it to add data for FlexGrid.

C#	copyCode
<pre>public class SalesDetails {     public SalesDetails(string companyName, double salesValue, double weighted,string salesStage)     {         CompanyName = companyName;         SalesValue = salesValue;         Weighted = weighted;         SalesStage = salesStage;     }     public SalesDetails() { }     public string CompanyName { get; set; }</pre>	

```

public double SalesValue { get; set; }
public double Weighted { get; set; }
public string SalesStage { get; set; }

public List<SalesDetails> GetData()
{
    string[] companyNames = { "Agilent Technologies", "Belo Co.", "Calpine
Co.", "Crompton Corp.", "Exelon Inc.", "Delphi Corp.", "Ferro Co.", "Gateway
Inc.", "Harris Corp." };
    string[] salesStages = { "Qualified", "Lead", "Qualified", "Proposal",
"Negotiation", "Won", "Lost", "Lead", "Proposal" };
    List<SalesDetails> salesDetailsList = new List<SalesDetails>();
    Random random = new Random();
    for(int i=0; i<9; i++)
    {
        salesDetailsList.Add(new
SalesDetails(companyNames[i], random.Next(0, 10000),
random.Next(1, 10000), salesStages[i]));
    }
    return salesDetailsList;
}
}

```

15. Add the following code to Form1's Load event to populate the grid with data.

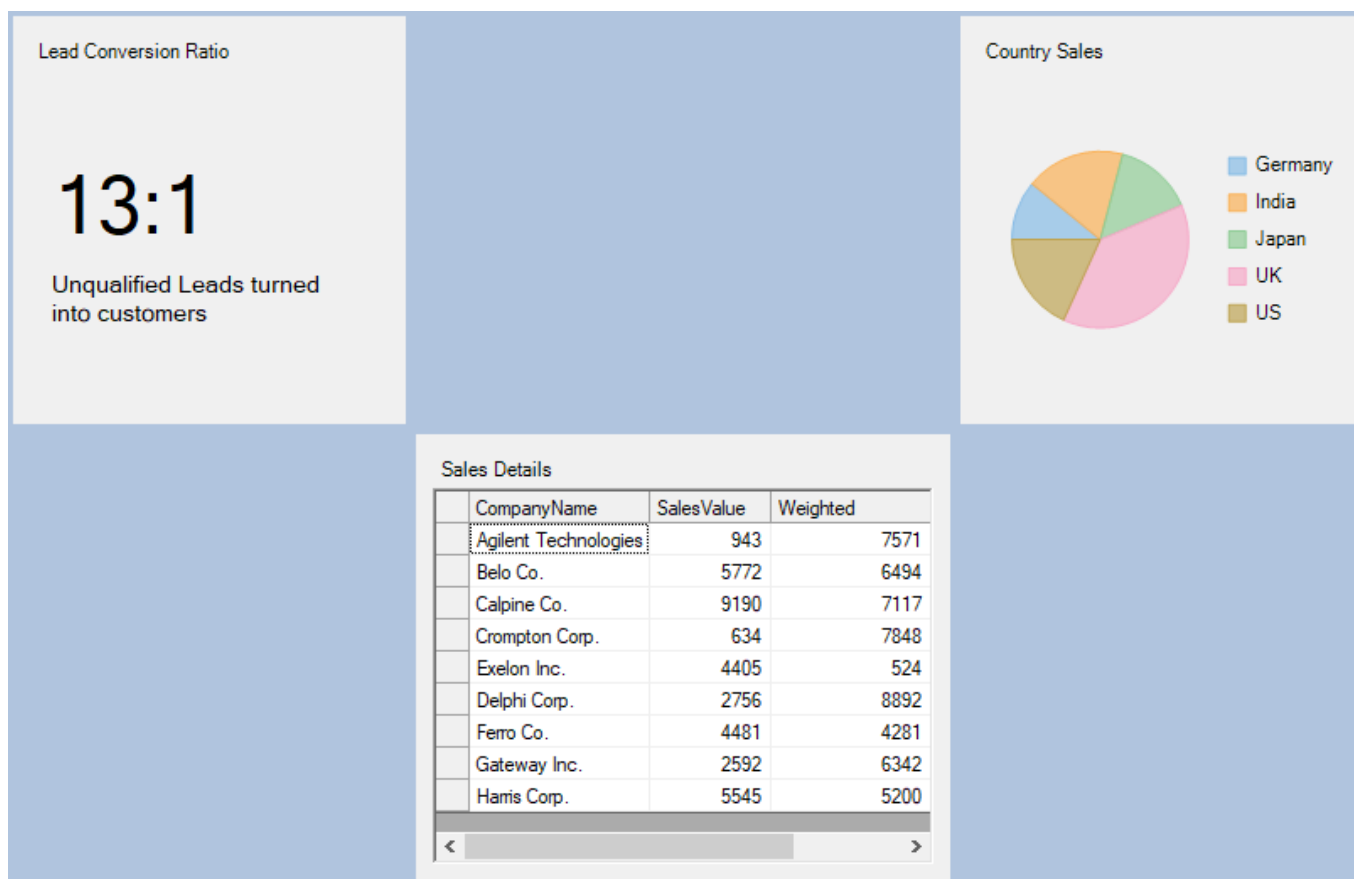
C#	copyCode
<pre> SalesDetails salesDetails = new SalesDetails(); List&lt;SalesDetails&gt; salesDetailsList = salesDetails.GetData(); c1FlexGrid1.DataSource = salesDetailsList; </pre>	

The lead conversion dashboard gets created.

## グリッドレイアウトでダッシュボードの作成

This topic guides you through the steps to create a lead conversion dashboard using grid layout. It displays the lead conversion ratio, sales in different countries in a pie chart, and sales details of companies in a FlexGrid.

# DashBoardLayout for WinForms



1. Create a new Windows Form application.
2. Drag and drop a DashboardLayout control from the Toolbox onto your form.  
Observe: By default, a layout of the type Split is attached to it.
3. Select the DashboardLayout control. In the **Properties** window, set the following properties:

Property Name	Value
LayoutType	Grid
Dock	Fill

On changing the LayoutType to Grid, the DashboardLayout control has six cells by default.

4. Drag and drop a Label control in the first cell and set the following properties:

Property Name	Value
Text	13:1
Font size	35

Observe: A child container named Panel1 of the type Panel is automatically created under the Label control.

5. Select **Panel1** and set the following properties:

Property Name	Value
Dock	Fill
Caption on c1DashboardLayout1	Lead Conversion Ratio

Now, drag and drop another Label control on Panel1 and set the following properties:

Property Name	Value
---------------	-------

Text	Unqualified Leads turned into customers
Font size	10

6. Drag and drop a FlexPie in the third cell and set the following properties:

Property Name	Value
Dock	Fill
BindingName	CountryName
Binding	OpportunityCount

A child container named Panel3 is created under it.

7. Select Panel3 and set the following properties:

Property Name	Value
Dock	Fill
Caption on c1DashboardLayout1	Country Sales

8. Add a class named CountrySales.cs to the project and copy the following code to it to add data for FlexPie.

C#	copyCode
<pre> public class CountrySales {     public CountrySales(string countryName, int opportunityCount)     {         CountryName = countryName;         OpportunityCount = opportunityCount;     }     public CountrySales() { }     public string CountryName { get; set; }     public int OpportunityCount { get; set; }     public List&lt;CountrySales&gt; GetData()     {         string[] countryNames = {"Germany", "India", "Japan", "UK", "US" };         List&lt;CountrySales&gt; countrySalesList = new List&lt;CountrySales&gt;();         Random random = new Random();         for (int i = 0; i &lt; 5; i++)             countrySalesList.Add(new CountrySales(countryNames[i],random.Next(0,30)));         return countrySalesList;     } } </pre>	

9. Add the following code to Form1's Load event to populate the pie with data.

C#	copyCode
<pre> CountrySales countrySales = new CountrySales(); List&lt;CountrySales&gt; countrySalesList = countrySales.GetData(); flexPie1.DataSource = countrySalesList; </pre>	

10. Drag and drop a FlexGrid control in the fifth cell.

A child container named Panel5 gets created under it.

11. Select Panel5 and set the following properties:

Property Name	Value
---------------	-------

## DashBoardLayout for WinForms

Dock	Fill
Caption on c1DashboardLayout1	Sales Details

12. Add a class named SalesDetails.cs to the project and copy the following code to it to add data for FlexGrid.

C#	copyCode
<pre>public class SalesDetails {     public SalesDetails(string companyName, double salesValue, double weighted, string salesStage)     {         CompanyName = companyName;         SalesValue = salesValue;         Weighted = weighted;         SalesStage = salesStage;     }     public SalesDetails() { }     public string CompanyName { get; set; }     public double SalesValue { get; set; }     public double Weighted { get; set; }     public string SalesStage { get; set; }      public List&lt;SalesDetails&gt; GetData()     {         string[] companyNames = { "Agilent Technologies", "Belo Co.", "Calpine Co.", "Crompton Corp.", "Exelon Inc.", "Delphi Corp.", "Ferro Co.", "Gateway Inc.", "Harris Corp." };         string[] salesStages = { "Qualified", "Lead", "Qualified", "Proposal", "Negotiation", "Won", "Lost", "Lead", "Proposal" };         List&lt;SalesDetails&gt; salesDetailsList = new List&lt;SalesDetails&gt;();         Random random = new Random();         for(int i=0; i&lt;9; i++)         {             salesDetailsList.Add(new SalesDetails(companyNames[i], random.Next(0, 10000), random.Next(1, 10000), salesStages[i]));         }         return salesDetailsList;     } }</pre>	

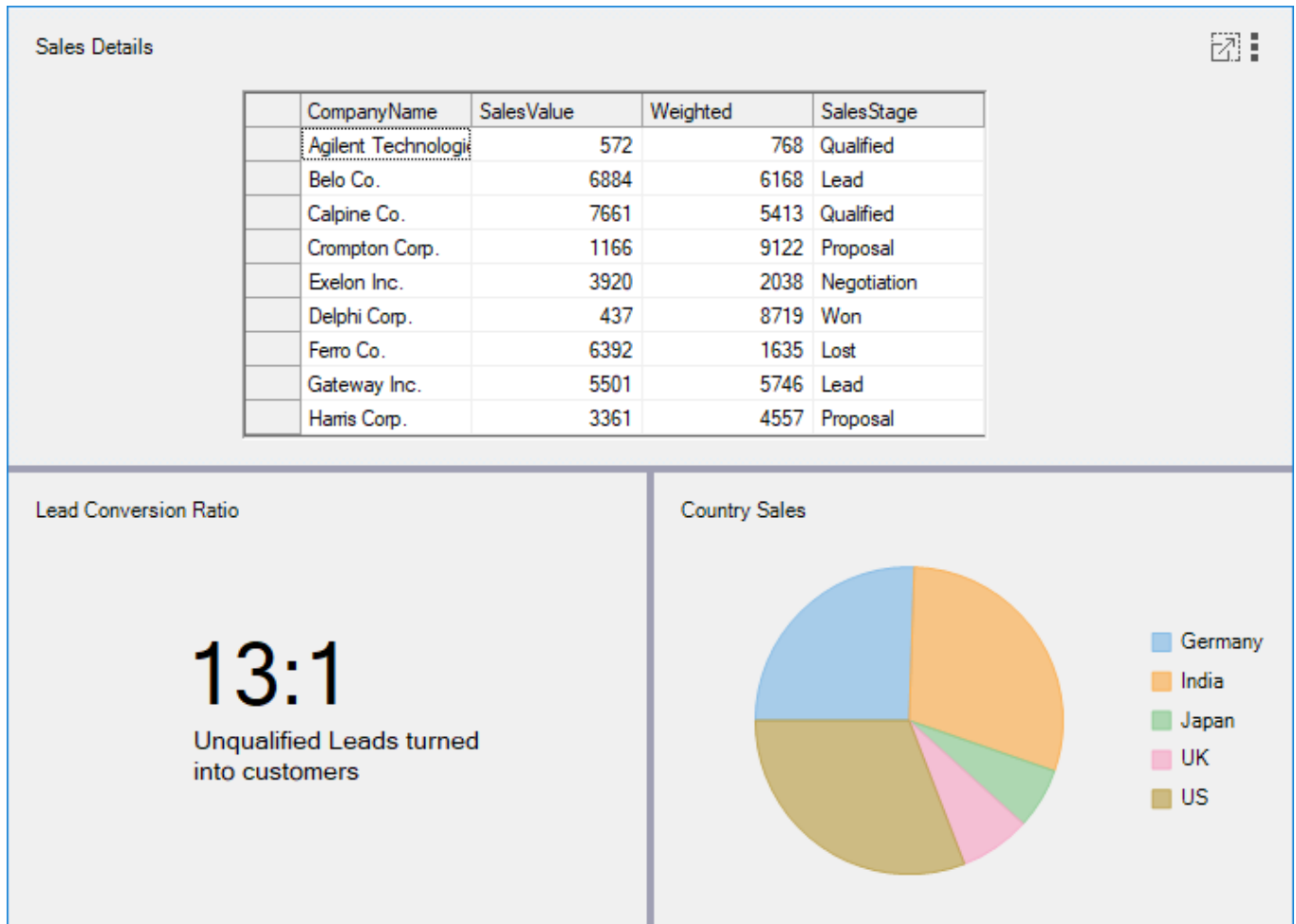
13. Add the following code to Form1's Load event to populate the grid with data.

C#	copyCode
<pre>SalesDetails salesDetails = new SalesDetails(); List&lt;SalesDetails&gt; salesDetailsList = salesDetails.GetData(); c1FlexGrid1.DataSource = salesDetailsList;</pre>	

The lead conversion dashboard gets created.

## 分割レイアウトでダッシュボードの作成

This topic guides you through the steps to create a lead conversion dashboard using split layout. It displays the lead conversion ratio, sales in different countries in a pie chart, and sales details of companies in a FlexGrid.



1. Create a new Windows Form application.
2. Drag and drop a DashboardLayout control from the Toolbox onto your form.  
Observe: By default, a layout of the type Split is attached to it.
3. Select the DashboardLayout control. In the **Properties** window, set its **Dock** property to **Fill**.
4. Click inside the DashboardLayout control. The SplitContentPanel (layout control attached to the DashboardLayout by default) is selected.

Note that when the DashboardLayout's LayoutType property is set to Split, child containers are not created automatically on dragging or dropping controls on the DashboardLayout as in other two layout types (Grid and Flow). Child containers have to be added manually to the DashboardLayout control.

5. Click on the SplitContentPanel's smart tag to open its Tasks Menu. Select Add Panel from the DashboardSplitContainer Tasks menu.  
A child container of the type C1SplitterPanel is added to the DashboardLayout control.
6. Now add two more panels using the Add Panel option from the DashboardSplitContainer Tasks menu.  
Observe: All the three panels namely C1SplitterPanel1, C1SplitterPanel2 and C1Splitterpanel3 are aligned one below the other horizontally.
7. Select C1SplitterPanel2 and set its **Dock** property to **Left**.
8. Drag and drop a FlexGrid control on C1SplitterPanel1. Set C1SplitterPanel1's Text property to Sales Details.  
This sets the text for C1SplitterPanel header.
9. Add a class named SalesDetails.cs to the project and copy the following code to it to add data for FlexGrid.

C# copyCode

```
public class SalesDetails
{
```

```

    public SalesDetails(string companyName, double salesValue, double
weighted,string salesStage)
    {
        CompanyName = companyName;
        SalesValue = salesValue;
        Weighted = weighted;
        SalesStage = salesStage;
    }
    public SalesDetails() { }
    public string CompanyName { get; set; }
    public double SalesValue { get; set; }
    public double Weighted { get; set; }
    public string SalesStage { get; set; }

    public List<SalesDetails> GetData()
    {
        string[] companyNames = { "Agilent Technologies", "Belo Co.", "Calpine
Co.", "Crompton Corp.", "Exelon Inc.", "Delphi Corp.", "Ferro Co.", "Gateway
Inc.", "Harris Corp." };
        string[] salesStages = { "Qualified", "Lead", "Qualified", "Proposal",
"Negotiation", "Won", "Lost", "Lead", "Proposal" };
        List<SalesDetails> salesDetailsList = new List<SalesDetails>();
        Random random = new Random();
        for(int i=0;i<9;i++)
        {
            salesDetailsList.Add(new
SalesDetails(companyNames[i], random.Next(0,10000),
random.Next(1,10000), salesStages[i]));
        }
        return salesDetailsList;
    }
}

```

10. Add the following code to Form1's Load event to populate the grid with data.

C#	copyCode
<pre> SalesDetails salesDetails = new SalesDetails(); List&lt;SalesDetails&gt; salesDetailsList = salesDetails.GetData(); c1FlexGrid1.DataSource = salesDetailsList; </pre>	

11. Drag and drop a Label control on C1SplitterPanel2 and set the following properties:

Property Name	Value
Text	13:1
Font size	35

Now, drag and drop another Label control on C1SplitterPanel2 and set the following properties:

Property Name	Value
Text	Unqualified Leads turned into customers
Font size	10

12. Add a FlexPie to C1SplitterPanel3 and set the following properties:

Property Name	Value
Dock	Fill
BindingName	CountryName
Binding	OpportunityCount

13. Set C1SplitterPanel3's Text property to "Country Sales"
14. Add a class named CountrySales.cs to the project and copy the following code to it to add data for FlexPie.

C#	copyCode
<pre> public class CountrySales {     public CountrySales(string countryName, int opportunityCount)     {         CountryName = countryName;         OpportunityCount = opportunityCount;     }     public CountrySales() { }     public string CountryName { get; set; }     public int OpportunityCount { get; set; }     public List&lt;CountrySales&gt; GetData()     {         string[] countryNames = { "Germany", "India", "Japan", "UK", "US" };         List&lt;CountrySales&gt; countrySalesList = new List&lt;CountrySales&gt;();         Random random = new Random();         for (int i = 0; i &lt; 5; i++)             countrySalesList.Add(new CountrySales(countryNames[i], random.Next(0, 30)));         return countrySalesList;     } } </pre>	

15. Add the following code to Form1's Load event to populate the pie with data.

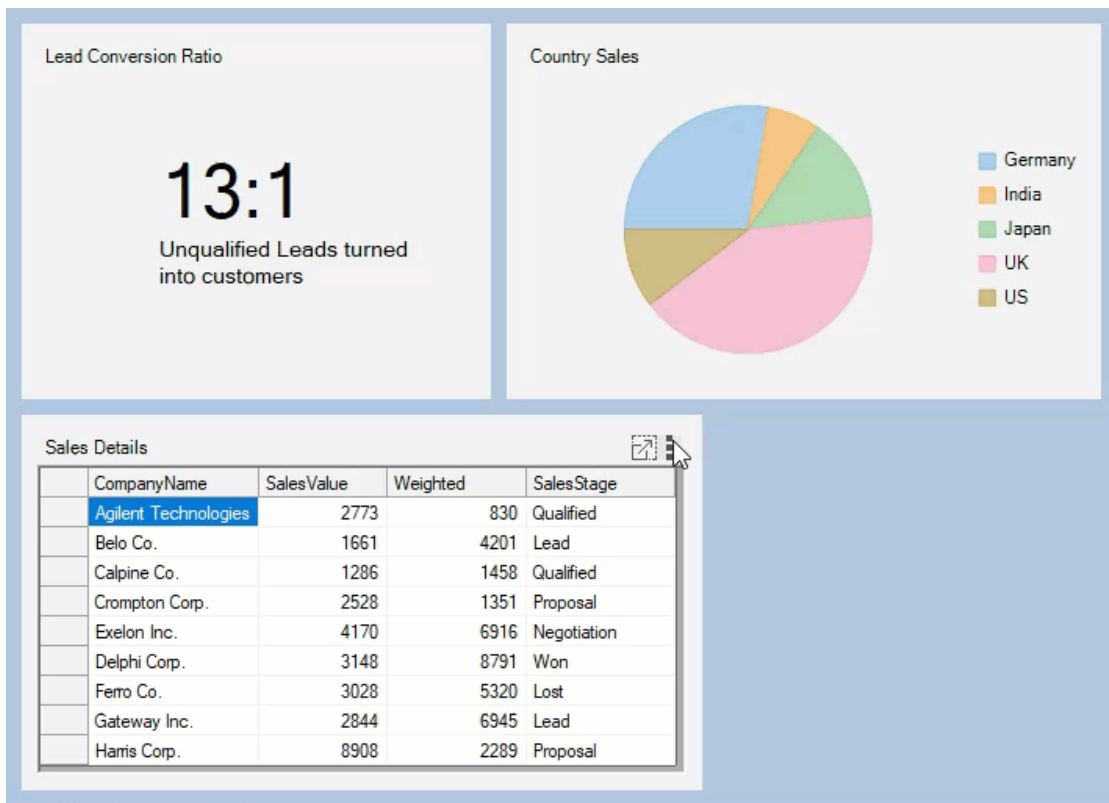
C#	copyCode
<pre> CountrySales countrySales = new CountrySales(); List&lt;CountrySales&gt; countrySalesList = countrySales.GetData(); flexPie1.DataSource = countrySalesList; </pre>	

The lead conversion dashboard gets created.

## カスタムコンテキストメニューの作成

This topic guides you through the steps to create custom context menu that provides you the options to change caption of the item container, export control, change image, and remove the item container from the DashboardCollection.

# DashBoardLayout for WinForms



1. Create a new Windows Forms App.
2. Drag and drop a **DashBoardLayout** control from the **Toolbox** onto your form.  
Observe: By default, a layout of the type Split is attached to it.
3. Select the DashboardLayout control. In the **Properties** window, set the following properties:

Property Name	Value
LayoutType	Flow
Dock	Fill

4. Click inside the DashboardLayout control. The FlowContentPanel (layout control attached to the DashboardLayout) is selected.
5. Drag and drop the **FlexGrid** control on the DashboardLayout and set the following properties:

Property Name	Value
Caption on c1DashboardLayout	Sales
DataSource	tenMostExpensiveProductsBindingSource

To create the data source and bind it to the control, follow the steps provided in the [クイックスタート](#).

6. Drag and drop the **FlexChart** control on the DashboardLayout and set the following properties:

Property Name	Value
Dock	Fill
DataSource	tenMostExpensiveProductsBindingSource
BindingX	TenMostExpensiveProducts
Binding	UnitPrice
Caption on c1DashboardLayout	Ten Most Expensive Products

7. Drag and drop a **PictureBox** control. In the **Properties** window, navigate to the **Image** property and click the ellipsis button to open the **Select Resource** dialog.

In the **Select Resource** window, click **Import** button to import an image from your system and click **OK**. Now, set the following properties of pictureBox1:

Property Name	Value
SizeMode	StretchImage
Size	374, 225
Caption on c1DashboardLayout	Logo

8. Drag and drop the **ContextMenuStrip** control, set its **Name** property to **customContextMenu**. Now, add the following menu items to the ContextMenuStrip:
  - o Change Caption
  - o Export
  - o Change Image
  - o Remove

Observe: The ContextMenuStrip control gets added to the Component tray.

Click the ContextMenuStrip smart tag and select **Edit Items**. The **Items Collection Editor** opens.

Observe the following members corresponding to the menu items you added are automatically added to the list:

- o changeCaptionToolStripMenuItem
- o exportToolStripMenuItem
- o changeImageToolStripMenuItem
- o removeToolStripMenuItem

9. Switch to the code view and add the following code:

C#	copyCode
<pre>private void CustomContextMenu_Opening(object sender, System.ComponentModel.CancelEventArgs e) {     //Change Image and Export options need to be shown selectively so they are disabled initially.     changeImageToolStripMenuItem.Enabled = false;     exportToolStripMenuItem.Enabled = false;      //Depending on the type of control contained within the child container     //the Export and Change Image     //options are enabled     foreach (Control control in clDashboardLayout1.GetSelectedItem().Items)     {         //If the selected child container contains a Flexgrid or FlexChart         //the export option is enabled.         if (control is ClFlexGrid    control is FlexChart)         {             exportToolStripMenuItem.Enabled = true;         }          //If the selected child container contains a PictureBox control         //the change image option is enabled.         if (control is PictureBox)         {             changeImageToolStripMenuItem.Enabled = true;         }     } }  private void ToolStripMenuItem_Click(object sender, EventArgs e) {     ToolStripMenuItem item = (ToolStripMenuItem)sender;     switch (item.Name)     {         case "changeCaptionToolStripMenuItem":              //Creates a new form which takes new Caption of the item container as input from the user.             CaptionChangeForm captionChangeForm = new CaptionChangeForm(                 clDashboardLayout1.GetSelectedItem().Caption);             if (captionChangeForm.ShowDialog() == DialogResult.OK)             {                 clDashboardLayout1.SetCaption(clDashboardLayout1.GetSelectedItem().ItemContainer,                     captionChangeForm.NewCaption);                 captionChangeForm.Dispose();             }             break;          case "exportToolStripMenuItem":             //Exports the FlexGrid to an Excel file and FlexChart to an image             //if the selected item container contains a FlexGrid or FlexChart</pre>	

# DashBoardLayout for WinForms

```
foreach (Control control in clDashboardLayout1.GetSelectedItem().Items)
{
    if (control is ClFlexGrid)
    {
        var dialog = new SaveFileDialog()
        {
            Filter = "Excel Files|*.xls;*.xlsx;*.xlsm"
        };

        if (dialog.ShowDialog() == DialogResult.OK)
        {
            ((ClFlexGrid)control).SaveExcel(dialog.FileName, FileFlags.AsDisplayed);
        }
    }
    else if (control is FlexChart)
    {
        var dialog = new SaveFileDialog()
        {
            Filter = "PNG|*.png|JPEG |*.jpeg|SVG|*.svg"
        };
        if (dialog.ShowDialog() == DialogResult.OK)
        {
            using (Stream stream = dialog.OpenFile())
            {
                var extension = dialog.FileName.Split('.')[1];
                ImageFormat fmt = (ImageFormat)Enum.Parse(typeof(ImageFormat), extension,
true);

                ((FlexChart)control).SaveImage(stream, fmt, 500, 800);
            }
        }
    }
}
break;
case "changeImageToolStripMenuItem":

    //Allows users to browse and change image file which is shown in the PictureBox
    //if the selected item container contains a PictureBox control
    foreach (Control control in clDashboardLayout1.GetSelectedItem().Items)
    {
        if (control is PictureBox)
        {
            var imageDialog = new OpenFileDialog()
            {
                Filter = "PNG|*.png|JPEG |*.jpeg|SVG|*.svg",
                Title = "Browse Image files"
            };
            if (imageDialog.ShowDialog() == DialogResult.OK)
            {
                ((PictureBox)control).Image = Image.FromFile(imageDialog.FileName);
            }
        }
    }
    break;
case "removeToolStripMenuItem":

    //Removes the selected item container from the DashboardCollection
    clDashboardLayout1.Items.Remove(clDashboardLayout1.GetSelectedItem().Id);
    break;
}
}
private void Form1_Load(object sender, EventArgs e)
{
    // TODO: This line of code loads data into the 'clNWindDataSet.Ten_Most_Expensive_Products'
    //table. You can move, or remove it, as needed.
```

```
this.ten_Most_Expensive_ProductsTableAdapter.Fill(this.clNWindDataSet.Ten_Most_Expensive_Products);

clDashboardLayout1.Options.ContextMenuStrip = customContextMenu;
customContextMenu.Opening += CustomContextMenu_Opening;

for (int i = 0; i < customContextMenu.Items.Count; i++)
{
    customContextMenu.Items[i].Click += ToolStripMenuItem_Click;
}
}
```

10. Add a new **Form** to the project and name it **CaptionChangeForm**. Add the following controls to the form and set their properties:

Control	Property	Value
Label	Text	Caption
TextBox	Name	captionTextBox
Button	Name	saveButton
Button	Text	Save

11. Switch to the code view and add the following code:

C# copyCode

```
public partial class CaptionChangeForm : Form
{
    //NewCaption is a public property which is accessed in Form1
    //to change the child container's Caption
    public string NewCaption { get; set; }
    public CaptionChangeForm(string oldCaption)
    {
        InitializeComponent();
        captionTextBox.Text = oldCaption;
    }

    private void saveButton_Click(object sender, EventArgs e)
    {
        //Sets the value of NewCaption property and closes the form
        if (captionTextBox.Text != String.Empty)
        {
            NewCaption = captionTextBox.Text;
            this.DialogResult = DialogResult.OK;
        }
        this.Close();
    }
}
```

A dashboard gets created wherein you can use the custom context menu.

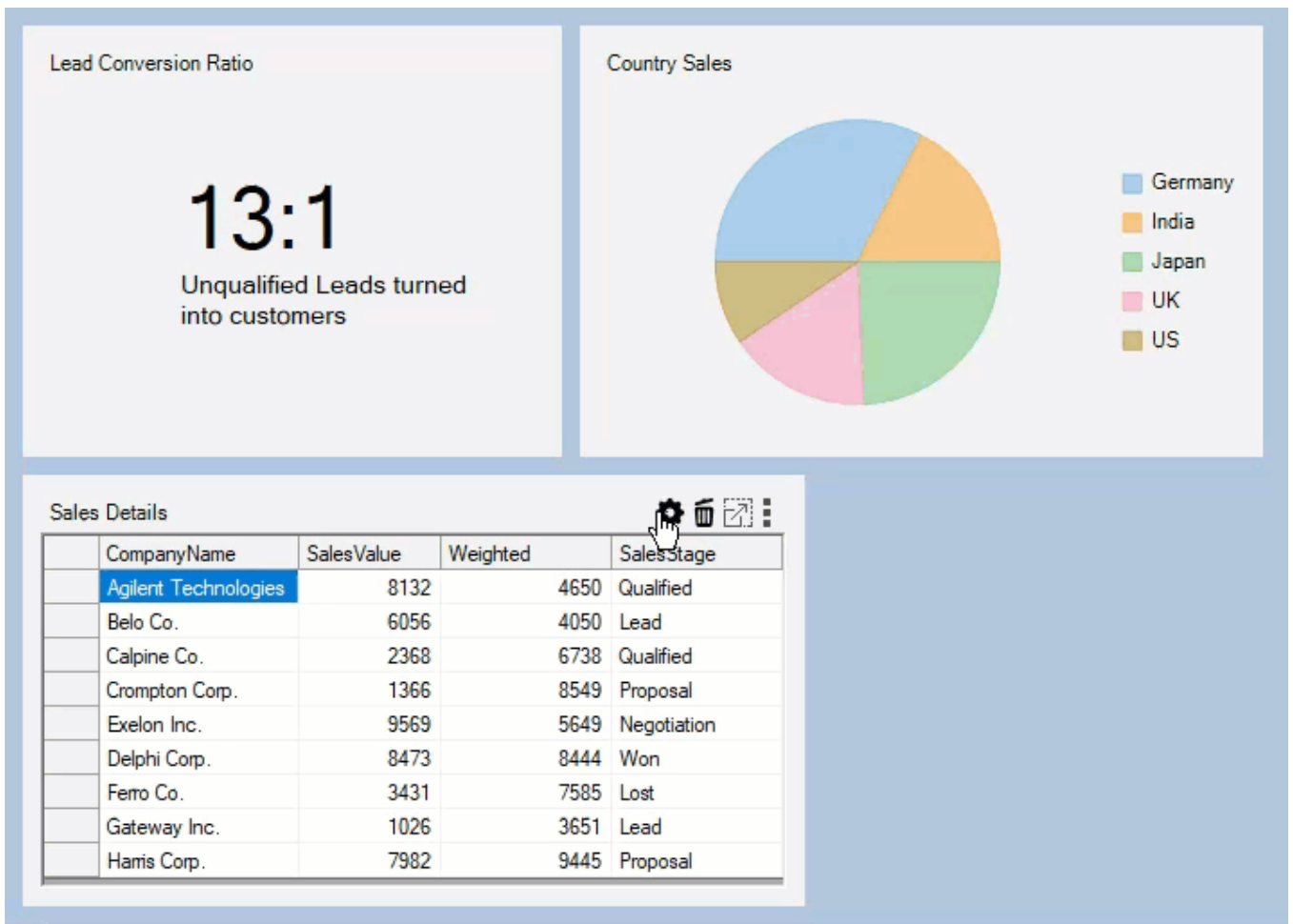
## 子コンテナのヘッダのカスタマイズ

This walkthrough explains the steps to add custom elements to the child container headers of the DashboardLayout control. As an example, this walkthrough helps you to add custom header elements to perform the following functions on the dashboard tiles:



- Remove a child container.
- Export the FlexPie or FlexGrid placed in the child container to Image or Excel respectively.
- Change the caption of the child container.

The following GIF image shows the custom header for each tile in the DashboardLayout control.

# DashBoardLayout for WinForms



1. Create a dashboard application using the [フローレイアウトでダッシュボードの作成](#) walkthrough topic.
2. In the Form's designer file, open the **C1DashboardLayout Tasks** menu by clicking on the smart tag glyph on the upper-right corner of the DashboardLayout control.
3. In the **C1DashboardLayout Tasks** menu, click on **Show Options.HeaderElements** panel. This opens a panel on the top side of the DashboardLayout control with the following text:  
'Drag control here to add it to Options.HeaderElements'.

 **Note:** The controls dragged and dropped in the Options.HeaderElements panel appear in the child container headers next to the default header elements. It is also possible to show/hide the Options.HeaderElements panel by using the downward/upward arrow  button which appears when the DashboardLayout control is selected in the designer file.

4. Drag and drop three **IconBox** controls from the **ToolBox** onto the **Options.HeaderElements** panel to add any WinForms controls directly to the panel. In this walkthrough, we are adding the IconBox controls as we need to display only three different icons which when clicked perform the desired actions.
5. In the **Properties** window, navigate to the **BackgroundImage** property of the IconBox controls, click the ellipsis button to open the **Select Resource** dialog.
6. In the **Select Resource** dialog, click **Import** button to import images for remove, export and settings icons from your system, and then click **OK**.
7. Subscribe to the click event for the added icon boxes to generate their respective click event handlers.
8. Add the following code in the **Click** event handler of the icon box with the remove icon to remove the item container from DashboardLayout.

```
C# copyCode  
  
private void removeIconBox_Click(object sender, EventArgs e)  
{  
    //DashboardLayoutからアイテムコンテナを削除します
```

```
c1DashboardLayout1.Items.Remove(c1DashboardLayout1.GetSelectedItem().Id);
}
```

## VB

```
Private Sub removeIconButton_Click(ByVal sender As Object, ByVal e As EventArgs)
    ' DashboardLayoutからアイテムコンテナを削除します
    c1DashboardLayout1.Items.Remove(c1DashboardLayout1.GetSelectedItem().Id)
End Sub
```

9. Use the following code to add a new form which acts as a popup window for renaming the caption that includes a TextBox control showing the old Caption and allows user to change that caption by clicking on the Save button.

## C#

copyCode

```
public partial class CaptionChangeForm : Form
{
    // 'NewCaptionは、子コンテナのキャプションを変更するためにForm1でアクセスされるパブリックプロパティです。
    public string NewCaption { get; set; }
    public CaptionChangeForm(string oldCaption)
    {
        InitializeComponent();
        captionTextBox.Text = oldCaption;
    }

    private void saveButton_Click(object sender, EventArgs e)
    {
        // NewCaptionプロパティの値を設定し、フォームを閉じます
        if (captionTextBox.Text != String.Empty)
        {
            NewCaption = captionTextBox.Text;
            this.DialogResult = DialogResult.OK;
        }
        this.Close();
    }
}
```

## VB

```
Public partial Class CaptionChangeForm
    Inherits Form
    ' NewCaptionは、子コンテナのキャプションを変更するためにForm1でアクセスされるパブリックプロパティです。
    Public Property NewCaption() As String
    End Property
    Public Sub New(ByVal oldCaption As String)
        InitializeComponent()
        captionTextBox.Text = oldCaption
    End Sub

    Private Sub saveButton_Click(ByVal sender As Object, ByVal e As EventArgs)
        ' NewCaptionプロパティの値を設定し、フォームを閉じます
        If captionTextBox.Text <> String.Empty Then
```

# DashBoardLayout for WinForms

```
        NewCaption = captionTextBox.Text
        Me.DialogResult = DialogResult.OK
    End If
    Me.Close()
End Sub
End Class
```

10. Add the following code to **Click** event handler of the icon box with the settings icon to change the caption of the child container.

C#	copyCode
<pre>private void changeCaptionIconBox_Click(object sender, EventArgs e) {     //ユーザーからの入力としてアイテムコンテナの新しいキャプションを受け取る新しいフォームを作成します     CaptionChangeForm captionChangeForm = new CaptionChangeForm(c1DashboardLayout1.GetSelectedItem().Caption);     if (captionChangeForm.ShowDialog() == DialogResult.OK)  c1DashboardLayout1.SetCaption(c1DashboardLayout1.GetSelectedItem().ItemContainer, captionChangeForm.NewCaption);     captionChangeForm.Dispose(); }</pre>	

VB
<pre>Private Sub changeCaptionIconBox_Click(ByVal sender As Object, ByVal e As EventArgs)     'ユーザーからの入力としてアイテムコンテナの新しいキャプションを受け取る新しいフォームを作成します     Dim captionChangeForm As CaptionChangeForm = New CaptionChangeForm(c1DashboardLayout1.GetSelectedItem().Caption)     If captionChangeForm.ShowDialog() = DialogResult.OK Then  c1DashboardLayout1.SetCaption(c1DashboardLayout1.GetSelectedItem().ItemContainer, captionChangeForm.NewCaption)     End If     captionChangeForm.Dispose() End Sub</pre>

11. Add the following code to **Click** event handler of the icon box with the export icon to export the control (i.e., FlexPie to image/FlexGrid to Excel) within the child container.

C#	copyCode
<pre>private void exportIconBox_Click(object sender, EventArgs e) {     //選択したアイテムコンテナにこれらのコントロールが含まれている場合は、     //FlexGridをExcelファイルにエクスポートし、FlexChartを画像にエクスポートします     foreach (Control control in c1DashboardLayout1.GetSelectedItem().Items)     {         var dialog = new SaveFileDialog();         if (control is C1FlexGrid)         {             dialog.Filter = "Excel Files *.xls;*.xlsx;*.xlsm";             if (dialog.ShowDialog() == DialogResult.OK)                 ((C1FlexGrid)control).SaveExcel(dialog.FileName, FileFlags.IncludeFixedCells);         }     } }</pre>	

```

    }
    else if (control is FlexPie)
    {
        dialog.Filter = "PNG|*.png|JPEG |*.jpeg|SVG|*.svg";
        if (dialog.ShowDialog() == DialogResult.OK)
        {
            using (Stream stream = dialog.OpenFile())
            {
                var extension = dialog.FileName.Split('.')[1];
                ImageFormat fmt =
                (ImageFormat)Enum.Parse(typeof(ImageFormat), extension, true);
                ((FlexPie)control).SaveImage(stream, fmt, control.Width,
                control.Height);
            }
        }
    }
}

```

## VB

```

Private Sub exportIconButton_Click(ByVal sender As Object, ByVal e As EventArgs)
    '選択したアイテムコンテナにこれらのコントロールが含まれている場合は、
    'FlexGridをExcelファイルにエクスポートし、FlexChartを画像にエクスポートします
    Dim control As Control
    For Each control In clDashboardLayout1.GetSelectedItem().Items
        Dim dialog As var = New SaveFileDialog()
        If TypeOf control Is ClFlexGrid Then
            dialog.Filter = "Excel Files|*.xls;*.xlsx;*.xlsm"
            If dialog.ShowDialog() = DialogResult.OK Then
                (CType(control, ClFlexGrid)).SaveExcel(dialog.FileName,
                FileFlags.IncludeFixedCells)
            End If
        Else If TypeOf control Is FlexPie Then
            dialog.Filter = "PNG|*.png|JPEG |*.jpeg|SVG|*.svg"
            If dialog.ShowDialog() = DialogResult.OK Then
                Imports (Stream stream = dialog.OpenFile())
                {
                    Dim extension As var = dialog.FileName.Split("."c)(1)
                    Dim fmt As ImageFormat =
                    CType(Enum.Parse(Type.GetType(ImageFormat), extension, True), ImageFormat)
                    (CType(control, FlexPie)).SaveImage(stream, fmt,
                    control.Width, control.Height)
                }
            End If
        End If
    Next
End Sub

```

- Subscribe to the **ItemContainerSelected** event in the **Form\_Load** event handler in order to enable the export option only for the tiles which contain the FlexGrid or FlexPie control.

## C#

copyCode

```
clDashboardLayout1.ItemContainerSelected +=
```

# DashBoardLayout for WinForms

```
C1DashboardLayout1_ItemContainerSelected;
```

VB

```
c1DashboardLayout1.ItemContainerSelected +=  
C1DashboardLayout1_ItemContainerSelected
```

Add the following code to the C1DashboardLayout\_ItemContainerSelected event handler.

C#

copyCode

```
private void C1DashboardLayout1_ItemContainerSelected(object sender, EventArgs e)  
{  
    exportIconBox.Visible = false;  
    foreach (Control control in c1DashboardLayout1.GetSelectedItem().Items)  
    {  
        //選択した子コンテナにFlexgridまたはFlexChartが含まれている場合、エクスポートオプションが  
        有効になります。  
        if (control is C1FlexGrid || control is FlexPie)  
            exportIconBox.Visible = true;  
    }  
}
```

VB

```
Private Sub C1DashboardLayout1_ItemContainerSelected(ByVal sender As Object,  
ByVal e As EventArgs)  
    exportIconBox.Visible = False  
    Dim control As Control  
    For Each control In c1DashboardLayout1.GetSelectedItem().Items  
        '選択した子コンテナにFlexgridまたはFlexChartが含まれている場合、エクスポートオプションが有  
        効になります。  
        If TypeOf control Is C1FlexGrid Or control Is FlexPie Then  
            exportIconBox.Visible = True  
        End If  
    Next  
End Sub
```

13. Run the application.

**Observe:** The child container headers show custom elements which can be used to perform different functions on the dashboard tiles.