

A Customer Quote Application in VB.NET

Illustrating the Control and DataTable Printing Features of PrintAdapters.NET

Overview

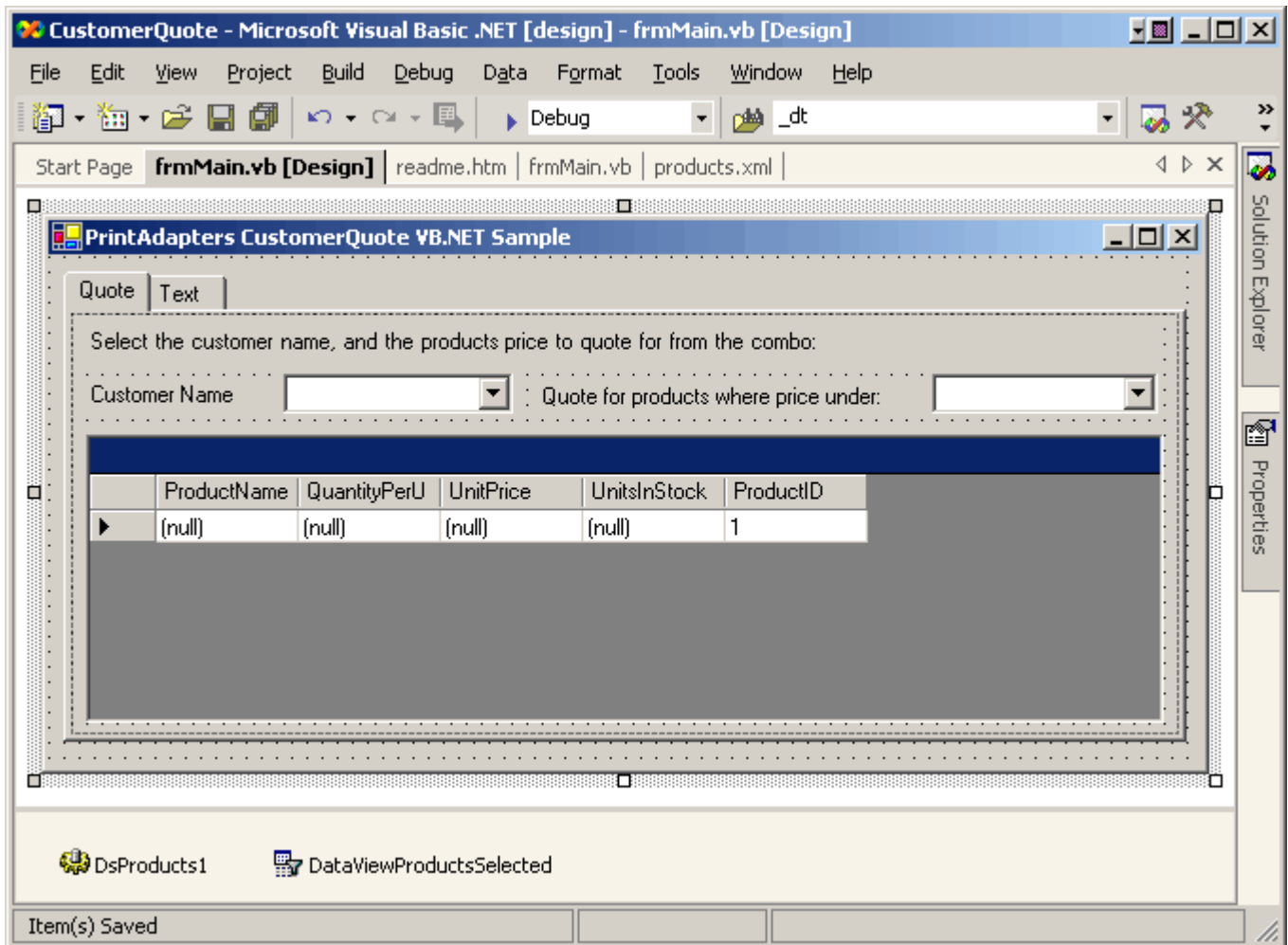
This sample demonstrates how to use **PrintAdapters** and **PrintChainManager** to create a finished application with the contents of multiple scrolling controls printed end to end to produce a printed customer quote.

It is quite common in an application to have a number of controls which display a large amount of data on screen by means of scrolling 'window' onto the data; typical examples include **ListView**, **TreeView**, **RichTextBox**, **DataGrid** etc. When it becomes necessary to print this data out, traditional approaches call for either: a) bespoke code to print the contents of the control over multiple pages, or b) the use of a reporting tool. Bespoke code is time consuming to write, and reporting tools frequently involve difficult configuration to achieve a similar effect - try printing **ListView**, **TreeView** or **RichTextBox** contents with your favourite reporting tool and you'll see what I mean.

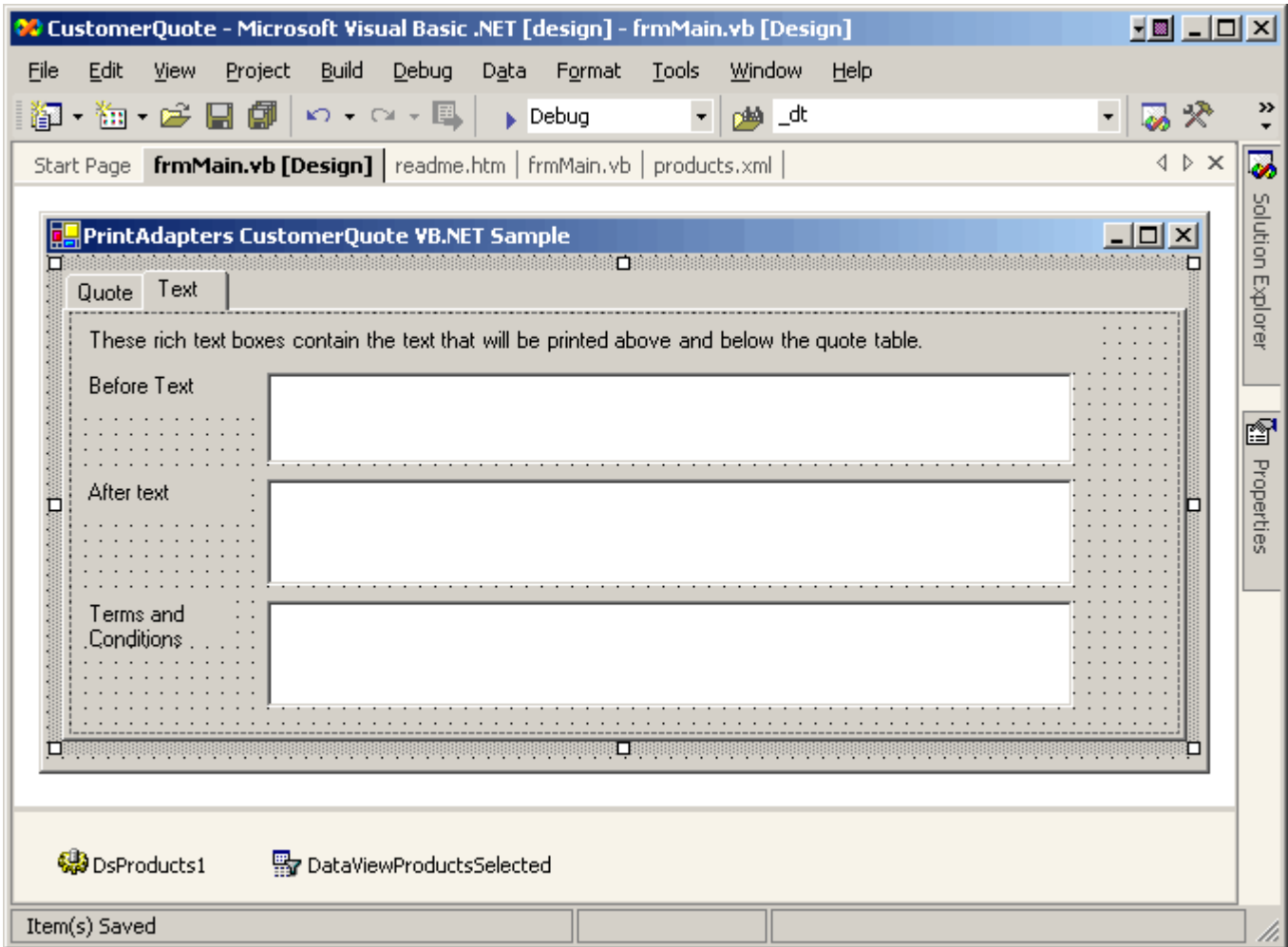
This is why we designed **PrintAdapters** for .NET. We wanted to be able to print the contents of controls such as **TreeView** and **ListView** without having to re-code everything to fit in with a reporting tool, and since .NET gives us such an extensible mechanism in the form of **PrintDocument**, **PrinterSettings** etc, a set of 'adapters' that render the data we wanted seemed the ideal way to go.

The .NET framework gives you the basic **PrintDocument**, and we have provided specialisations suited to printing each of the controls discussed above. We have also implemented a further **PrintDocument** based component **PrintChainManager** that stitches together the output of multiple **PrintDocument** derived classes to form a single print job, and handles the tracking of how much space on the page each **PrintDocument** has used, so that the next one can start printing where the previous one left off.

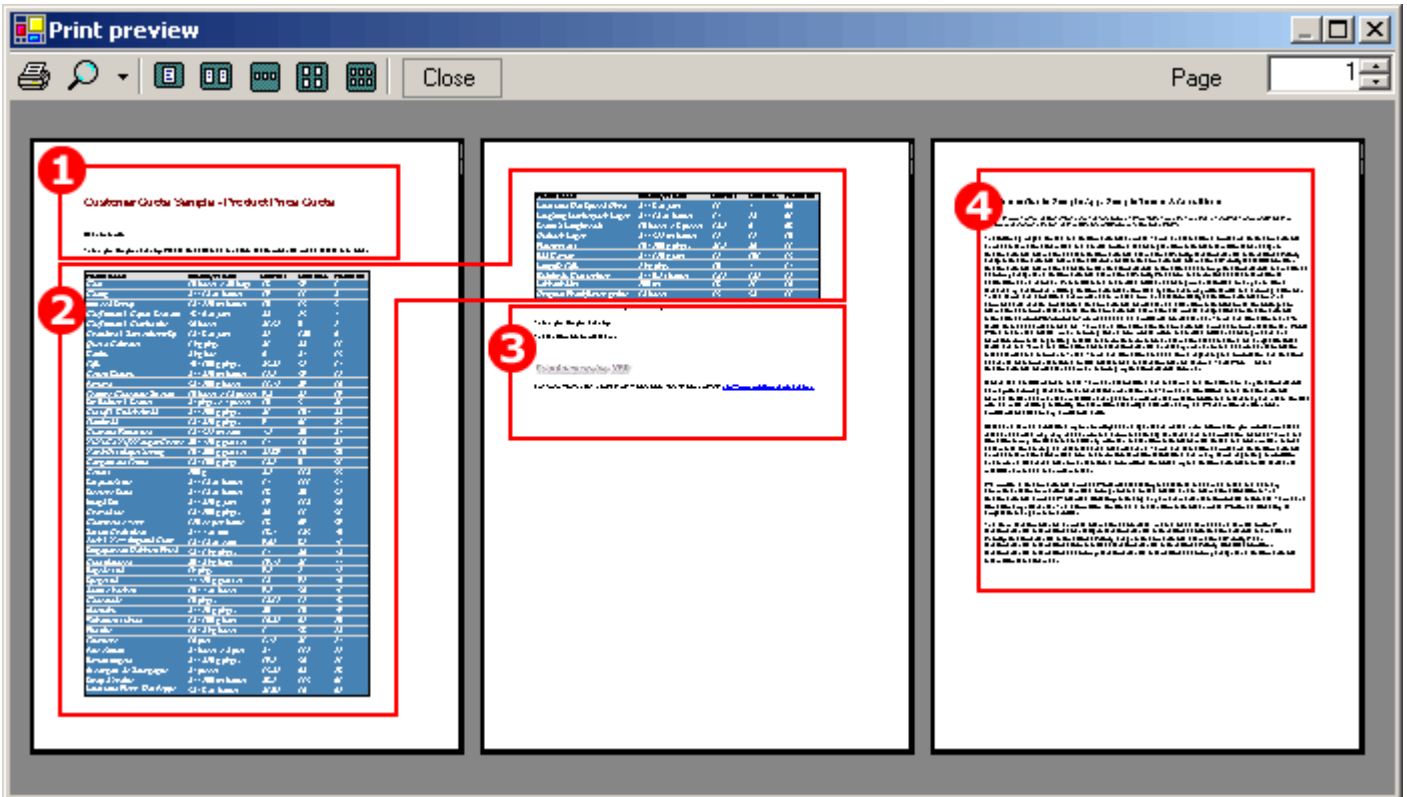
Application Architecture



The CustomerQuote application consists of a main form, frmMain. To this we have added a TabControl: the first tab holds the DataGrid of mock product data, and a number of controls that allow us to customise the output.



The second tab holds three RichTextBox controls. When we produce the customer quote printout we want the contents of these controls printed in this order: 1. Contents of RichTextBox1, 2. Contents of the DataGrid, 3. Contents of RichTextBox2, 4. Contents of RichTextBox3 - to produce something like this (Note that we specifically want the RichTextBox 4, the Terms & Conditions text, to appear on a new page on its own - we'll deal with how we do this later):



Setting up the DataSet

The mock product data is loaded from an embedded resource XML file into the DataSet DSProducts1 like this:

```
Me.DsProducts1.ReadXml([Assembly].GetExecutingAssembly().GetManifestResourceStream("CustomerQuote.products.xml"))
```

Note: the key point when loading embedded resources is to get the fully qualified name of the resource (the part in quotes in the above code line) exactly right. A handy tip for checking what you should use is to open the **ILDASM.exe** tool that comes with Visual Studio, point it at your compiled executable, and then double click on the **MANIFEST** section. This lists, among other things, the fully qualified names of all the embedded resources.

Filtering the contents of the DataGridView using a DataView

We've added a **DataView** (called **DataViewProductsSelected**) to **frmMain** as well, whose **DataSource** is **DSProducts1.Products** - this will allow us to filter the data displayed on the printout. **DataGrid1's DataSource** is set to this **DataView**.

We set the filter expression when the price combo's selection changes like this:

```
Me.DataViewProductsSelected.RowFilter = "UnitPrice < " & Me.cmbMinPrice.Text
```

Setting up the RichTextBox Controls

We've included the standard RTF text we want to appear in each of the RichTextBoxes as embedded resources in the project then we load them in with a call to **GetManifestResourceStream** as follows:

```
Me.rtbTC.LoadFile([Assembly].GetExecutingAssembly().GetManifestResourceStream("CustomerQuote.TermsAndConditions.rtf"), RichTextBoxStreamType.RichText)
```

Adding Printing Support

At this point we have a form with a **DataGrid** populated from a **DataView**. When we change the minimum price in the combo, the **DataView** is refreshed and the **DataGrid** reflects the change.

Now we need to produce the printout... this is actually quite easy with PrintAdapters, and is largely a case of dropping components onto the form and configuring their properties with point-and-click:

1. Add a `PrintRichTextBox` component for each of the `RichTextBox` controls, set each one's `RichTextBoxControl` property to the appropriate `RichTextBox` control
2. Add a `PrintDataTable` component to the form. Set its `DataSource` property to the products selection `DataView`
3. Add a `PrintChainManager` to the form
4. Now we need to add a small amount of code to tell the `PrintChainManager` about the sequence of documents it is to print. Add the following code to the form's `Load` event handler:

```
Me.PrintChainManager1.Documents.Add(Me.PrintRichTextBox1)
Me.PrintChainManager1.Documents.Add(Me.PrintDataTable1)
Me.PrintChainManager1.Documents.Add(Me.PrintRichTextBox2)
Me.PrintChainManager1.Documents.Add(Me.PrintRichTextBox3)
```

5. Add a standard .NET `PrintPreviewDialog` component to the main form. Set its `Document` property to the `PrintChainManager` reference
6. Now add a menu item click handler to call `PrintPreviewDialog.ShowDialog`:

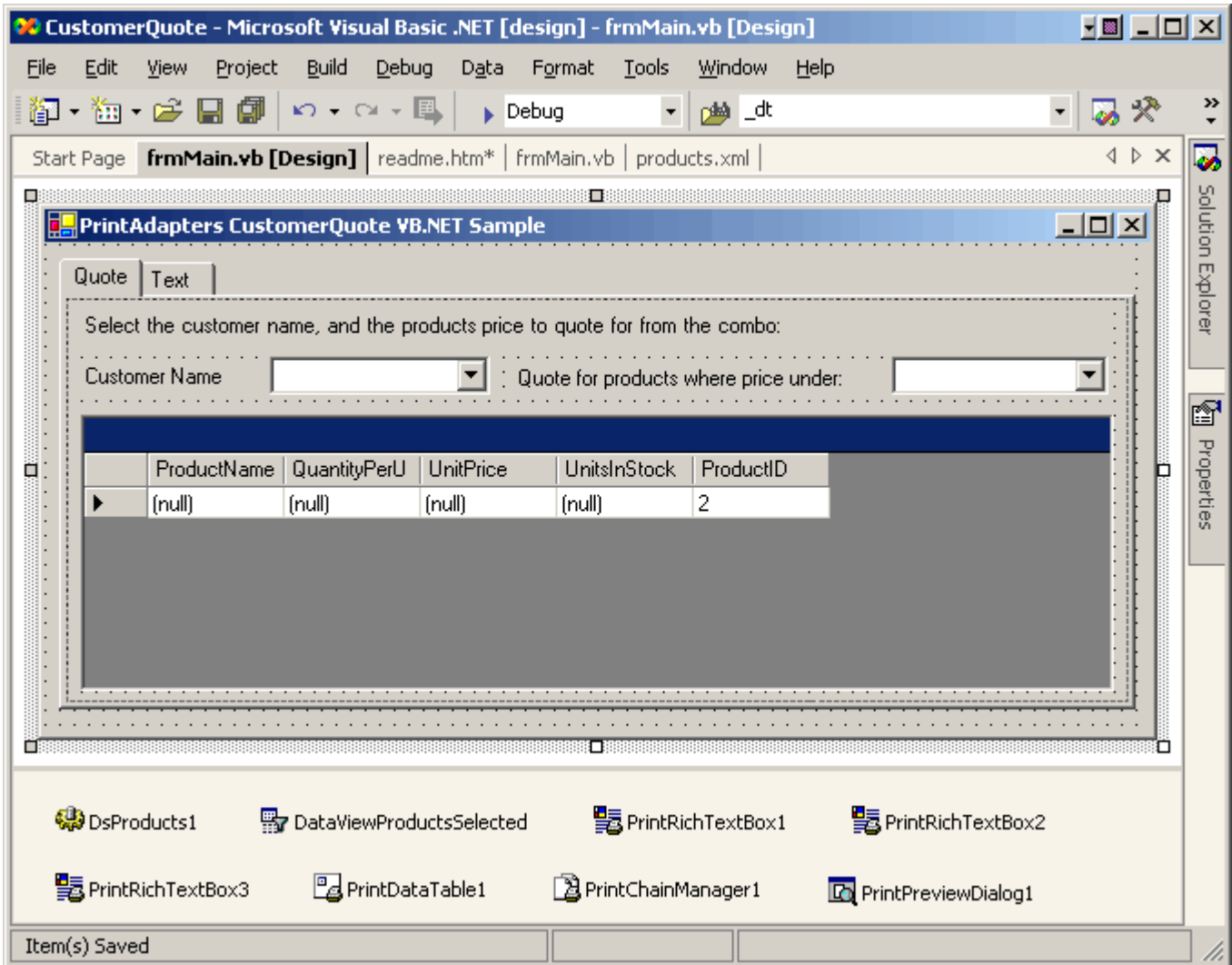
```
Me.PrintPreviewDialog1.ShowDialog()
```

Finally, we mentioned earlier that we want the contents of the final `RichTextBox`, holding the Terms & Conditions text, to appear on a new page rather than flowing on from the previous text. To do this we need to turn off the `FlowDocuments` property of `PrintChainManager`. This is set true by default, and in this state ensures that each new `PrintAdapter` starts printing immediately after the previous one on the same page. When it is set false each new `PrintAdapter` prints on a new page. So what we do is add a handler for the `PrintChainManager.DocumentChanged` event, and when we detect that the `ActiveDocument` is `RichTextBox3` turn off `FlowDocuments`, thus ensuring that `RichTextBox3` is printed on a new page:

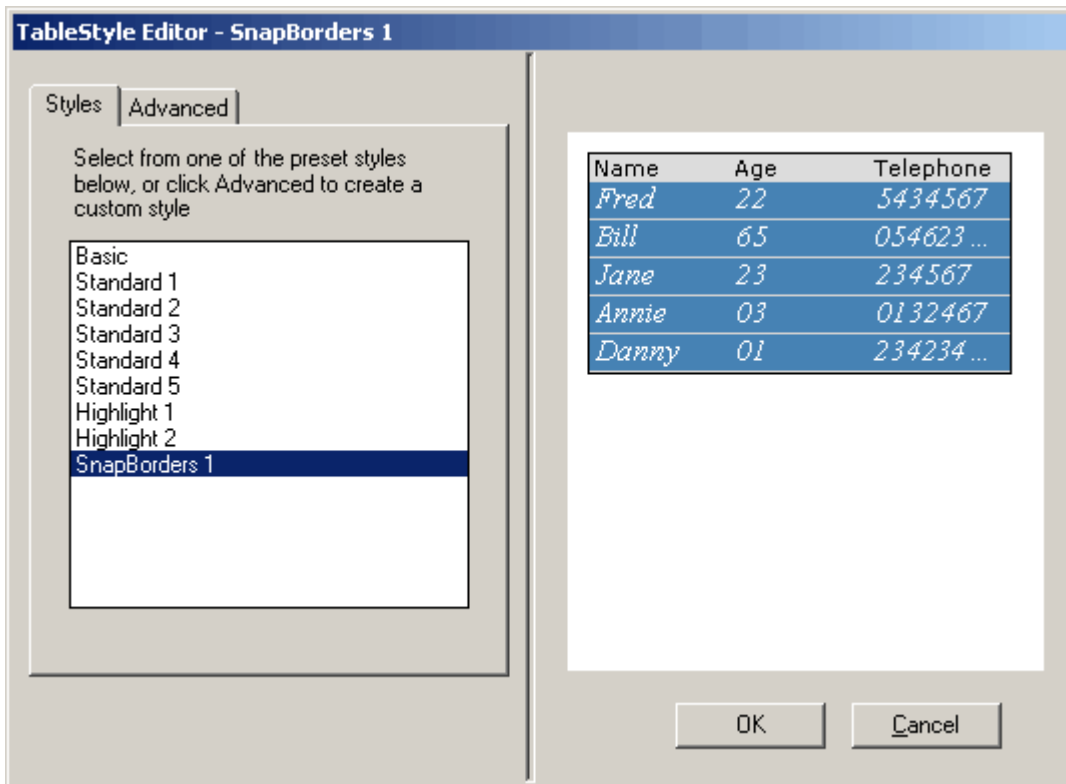
```
Private Sub PrintChainManager1_DocumentChanged(ByVal sender As Object, ByVal e As System.EventArgs) Handles
PrintChainManager1.DocumentChanged
    Me.PrintChainManager1.FlowDocuments = Not (Me.PrintChainManager1.ActiveDocument Is Me.PrintRichTextBox3)
End Sub
```

The Finished Product

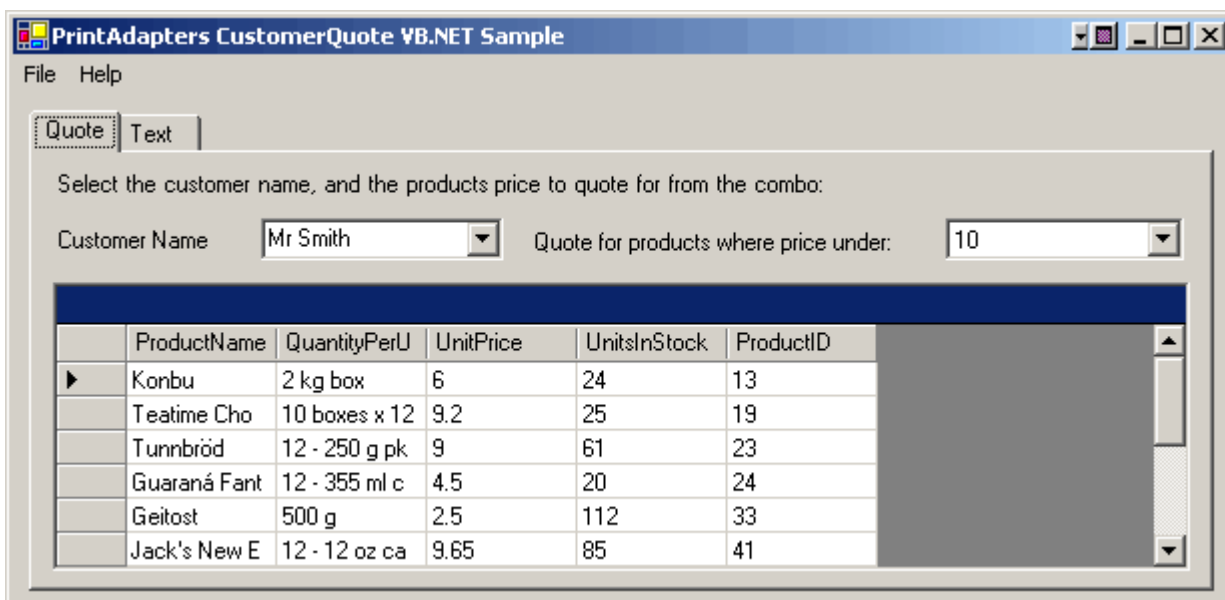
Now, in the design environment we have a form with a number of components in the component tray.



We set up a TableStyle for the PrintDataTable using the TableStyle designers (these are discussed in detail in the "TableStyle How To..." - see the help file):



Running the application:



And the corresponding print preview... because PrintAdapters renders the control contents in a resolution independent manner the result is a high quality printout, without any of the effort of writing all the printing code yourself:

