DZone Refcardz

CONTENTS INCLUDE:

Getting Started with Visual Studio 2010

By Alessandro Del Sole

INTRODUCTION

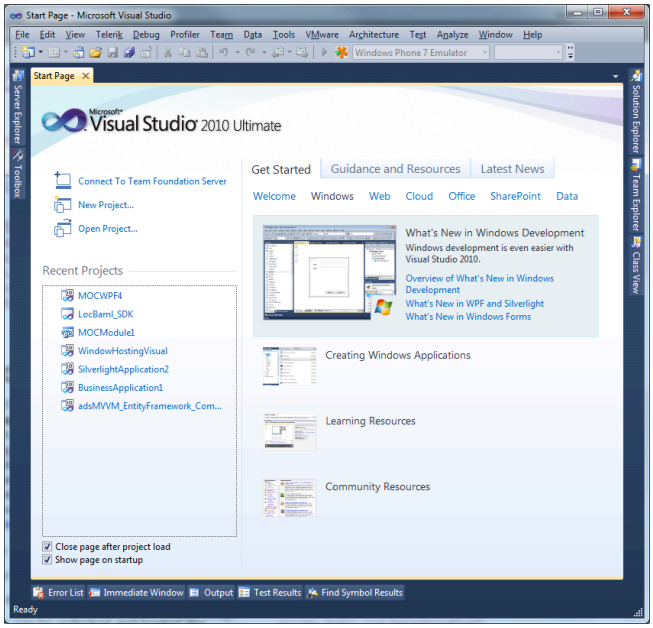
The new version of the Visual Studio 2010 Integrated Development Environment ships with a number of new important features. Such features are both related to the IDE architecture and to the coding experience, with the addition of new tools as well. This Refcard is a quick reference to the most important new features and instruments in Visual Studio 2010, also providing tips to most common shortcuts.

STARTING WITH VISUAL STUDIO 2010

Visual Studio 2010 is the developer tool used for creating .NET applications. There are different editions which cover every need, such as the Ultimate edition or the free Express editions (the latter are specific for hobbyists and students).

Starting the IDE

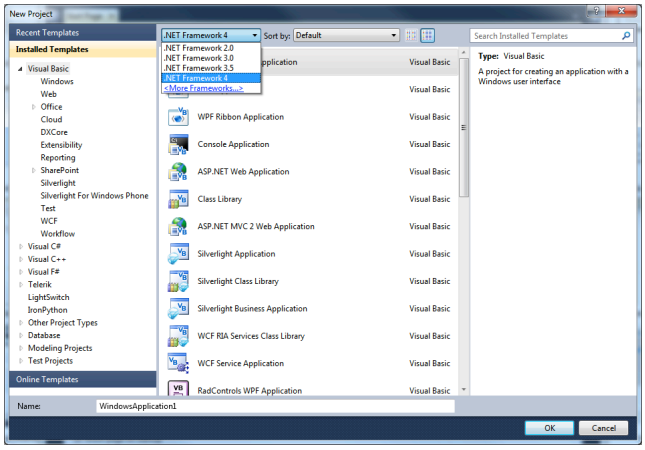
When you run Visual Studio 2010, the first thing you notice is a new Start Page which provides shortcuts, organized into tabs, to lots of learning resources about programming different Microsoft technologies with Visual Studio. These include the .NET Framework and programming languages, data platform, cloud computing or tools for teams. The following figure shows how the Start Page looks:



Visual Studio 2010 offers a convenient way for managing the most recently used projects by simply right-clicking each project name in the list. With this action you can pin or remove items from the list itself.

Creating applications

You create .NET applications in Visual Studio 2010 by first creating projects. A project is the place where all the parts composing your application, such as code, libraries, dialogs, pages and resources, reside. An application can be composed of one or more projects. Multiple projects bound together are combined into a Solution. Thus, the first step is creating a new project. You accomplish this by selecting the New|Project command from the File menu. This will show the New Project dialog which looks like this:



As you can see, there are a number of available project types, grouped by programming language or by typology. As you can see from the above figure, the programming language project types are sub-grouped by technology. Once you have selected the desired project type, just assign a name and click OK. The Visual Studio documentation that ships with the product provides full descriptions for both Windows and Web projects.

In Visual Studio 2010 you can also target different versions of the .NET Framework (as shown in the figure above) or search specific project templates using the Search Installed Templates search box.

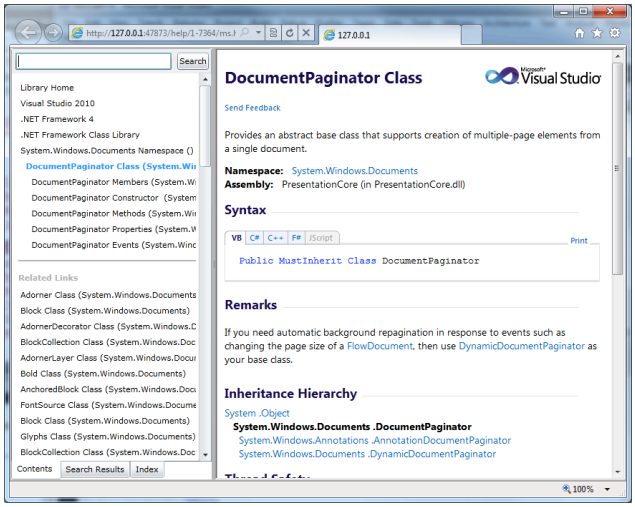
Building and debugging applications

You can build and run your executable application by pressing F5. This is common to both Windows and Web applications and will run your application with an instance of the Visual Studio debugger attached. Visual Studio provides a lot of debugging tools available in the Debug menu. For example, you can place breakpoints (also by pressing F9) or check for the value of local variables via the Locals tool window. If you don't want to run the application under the debugger investigation, simply press Shift+F5.

You can quickly deploy your application using the ClickOnce technology, which you access via the Build|Publish command.

Browsing the documentation

You can access the Visual Studio documentation in different situations by pressing F1. For example you can get general information by pressing F1 anywhere or get specific help for an element in your code by selecting the code element (such as the name of a type) and, again, press F1. In Visual Studio 2010 the help system has been rebuilt completely and now allows you to browse the documentation inside your Web browser. The following figure shows an example:

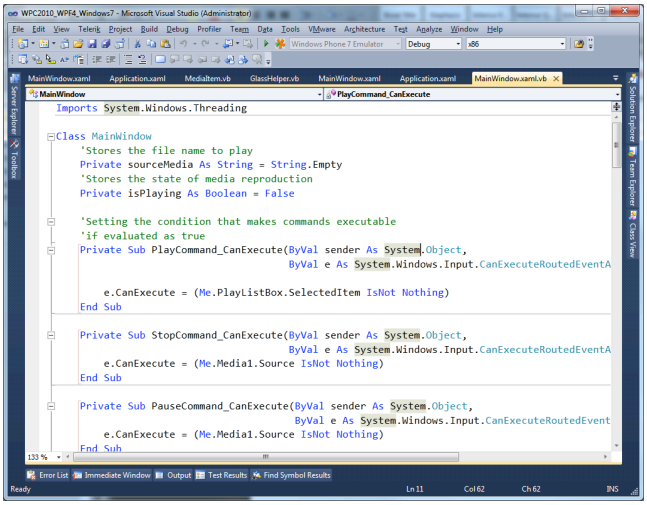


WPF-BASED CODE EDITOR

Most of the Visual Studio 2010 architecture now relies on Windows Presentation Foundation, the most recent technology from Microsoft for building rich client applications. If you are an existing Visual Studio developer this is something you immediately notice when running the IDE for the first time; it affects a number of areas such as the Start Page and tabbed dialogs or tool windows. The code editor is also built upon WPF which provides some great benefits, including:

- Better text rendering
- Code editor zoom by pressing CTRL + mouse wheel
- Automatic selection of all occurrences of a word
- Tabbed code dialogs can be undocked and treated like any other window outside the IDE
- **Improved extensibility:** the code editor also takes advantage of MEF (Managed Extensibility Framework) to receive pluggable components. This allows hosting rich content inside the code editor, such as images, videos, WPF controls, comment adornments and so on.
- **Reusability:** you can reuse the code editor as a component in your own Visual Studio add-ins.

You interact with the code editor via the IWPFTextView interface which is described in the MSDN Library. The following image shows how the WPF-based code editor looks:



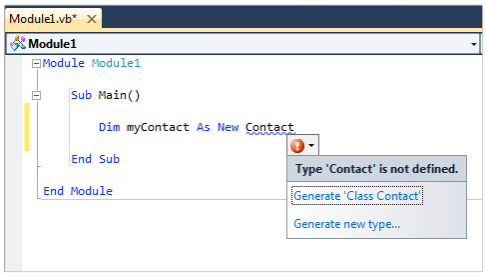
Notice that the zoom percentage is greater than 100%. This was accomplished simply by pressing CTRL + mouse wheel. A good number of code editor extensions are available in the Visual Studio Gallery Web site.

"GENERATE FROM USAGE" TOOL

In its continuous evolution, the Visual Studio environment has committed to offer even more sophisticated tools for helping developers write code. IntelliSense®, syntax highlighting, and auto-completion are just a few examples. It previously lacked the ability to create new objects on the fly. With Visual Studio 2010 you can now create new objects while writing code. This is useful when you realize that you need an object that you did not consider before or that you need to extend an existing object with new members. This functionality is integrated into the code editor and is known as Generate From Usage; it is available for both Visual Basic and C#.

Creating new types on the fly

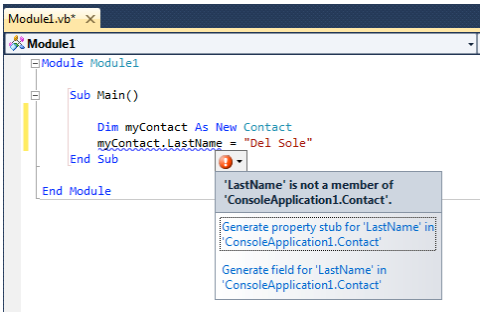
Let's start describing Generate From Usage by creating a new Console application. Consider the following situation, in which the code attempts to create an instance of the Contact class that actually does not exist yet and pay attention to the error correction options provided by the IDE:



Click Generate 'Class Contact' and Solution Explorer you will notice the presence of a file named Contact.vb (or .cs in C#) which contains the empty definition of the new class.

Hot Tip The class declaration gets the default visibility modifier, according to the current programming language, which is Friend for Visual Basic and Internal for C#. You need to change this manually if you want to expose the class outside the assembly.

Now you can populate your class with new members, like in the following figure, where the code attempts to assign a non-inexistent property, but the IDE provides the possibility of generating a property or a field:



If you select the first option, Visual Studio will add a property named LastName of type String to the Contact class. Similarly you can add both instances and shared methods on the fly. It is worth mentioning that Visual Studio:

- generates members of the appropriate type when possible.
- is able to generate interfaces and methods referring to delegates according to the current scenario. When generating such methods, it is able to add the appropriate signature.

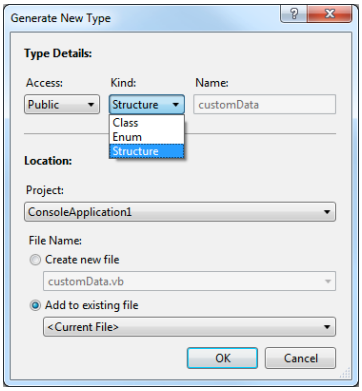
You saw how to create a class, which is the default type, and members with default visibility. But you can do more with Generate From Usage as explained in next subsection.

Creating different kinds of types

There are situations in which you need to create different kinds of objects, such as structures or enumerations. Generate From Usage makes this task easy. For example, you might want to create a new structure. If you type the following line of code:

```
Dim someData As New CustomData
```

the IDE marks CustomData as non-existent. From the error correction suggestions popup, simply select Generate New Type. This will prompt you with the Generate New Type dialog shown in the next figure:



Here you can:

- Provide the new object's visibility by setting the Access property
- Decide what kind of object you want to create from the Kind property
- Add the new member to a different project in your solution via the Project combo box
- Create a new code file or add the new object to an existing code file inside the File Name box

Simply click OK when you are done and the new object will be added to your solution according to the specified settings. At this point you can add new members as described in the previous section.

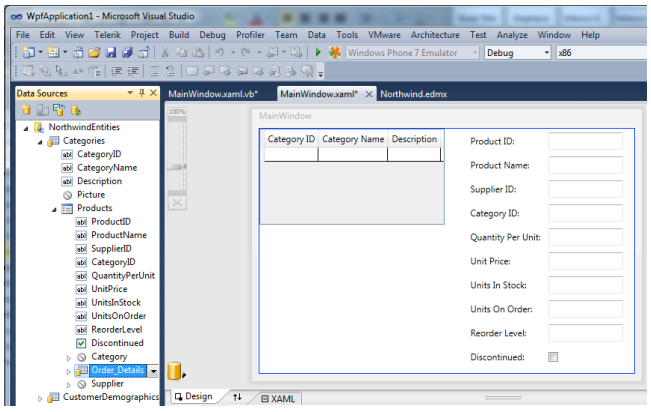
WPF & SILVERLIGHT: DRAG AND DROP DATA-BINDING

Visual Studio 2010 introduces an important feature for data development to both WPF and Silverlight: drag and drop data-binding. This technique has been very popular in the past among Windows Forms developers since it allows creating data forms quickly and easily. In order to make the data development experience easier in modern client technologies, such a technique is now available for WPF and Silverlight. In WPF full support is offered for DataSets and ADO.NET Entity Framework. With full support we intend complete UI and code-behind code generation. Support is also offered for custom business objects or different data sources such as LINQ to SQL or WCF Data Services, but only on the UI side, meaning that you need to write all the data access code from scratch. In Silverlight you don't have DataSets, so you can use this kind of data-binding with Entity Framework and custom objects.

Hot Tip The drag and drop data-binding is also available for WPF 3.5 and Silverlight 3 projects.

Creating the data-bound interface

Whatever your data source is, you may open the Data Sources window and then simply drag your desired data item from the window onto the designer surface, as it was done previously in Windows Forms applications. The following figure provides an example based on the Entity Framework as the data source:



Hot Tip The Data Sources window in Visual Studio 2010 provides full support for entities exposed by the Entity Framework.

As in older technologies, you can still create grid views or details views and you are allowed to replace the default controls with different ones. You can drag an entire object to create a view or a single object onto an items container control such as the ListBox for creating lists.

Hot Tip

Visual Studio adds controls to the user interface depending on the version of the .NET Framework you are targeting. For example, if you want to create a grid view, Visual Studio uses the DataGrid control in .NET 4 and a ListView control in .NET 3.5.

Dragging and dropping the data is just the first step since you obviously need some code.

A tour of the code

When you use the drag and drop binding, Visual Studio generates some useful code for you. This not only refers to the user interface and controls, but also to querying data. On the XAML side, it first generates the appropriate CollectionViewSource objects acting like "bridges" between data and UI:

```
    <Window.Resources>
        <CollectionViewSource x:Key="CategoriesViewSource" d:DesignSo
urce="{d:DesignInstance my:Category, CreateList=True}" />
        <CollectionViewSource x:Key="CategoriesProductsViewSo
urce" Source="{Binding Path=Products, Source={StaticResource
CategoriesViewSource}}" />
    </Window.Resources>
```

Next it generates data-bound controls, such as a DataGrid and Label/TextBox couples if considering the previous image (notice the Binding markup extension pointing to the bound property):

```
 <DataGrid AutoGenerateColumns="False" EnableRowVirtualization="True"
Height="128" HorizontalAlignment="Left" ItemsSource="{Binding}"
Name="CategoriesDataGrid" RowDetailsVisibilityMode="VisibleWhenSelect
ed" VerticalAlignment="Top" Width="248">
            <DataGrid.Columns>
                <DataGridTextColumn x:Name="CategoryIDColumn"
Binding="{Binding Path=CategoryID}" Header="Category ID"
Width="SizeToHeader" />
                <DataGridTextColumn x:Name="CategoryNameColumn"
Binding="{Binding Path=CategoryName}" Header="Category Name"
Width="SizeToHeader" />
                <DataGridTextColumn x:Name="DescriptionColumn"
Binding="{Binding Path=Description}" Header="Description"
Width="SizeToHeader" />
            </DataGrid.Columns>
        </DataGrid>
…
            <Label Content="Product ID:" Grid.Column="0" Grid.Row="0"
HorizontalAlignment="Left" Margin="3" VerticalAlignment="Center" />
            <TextBox Grid.Column="1" Grid.Row="0" Height="23"
HorizontalAlignment="Left" Margin="3" Name="ProductIDTextBox"
Text="{Binding Path=ProductID, Mode=TwoWay,
ValidatesOnExceptions=true, NotifyOnValidationError=true}"
VerticalAlignment="Center" Width="100" />
…
```

It is worth mentioning that Visual Studio also maps data types to the appropriate controls. A typical example is when you have data of type System.DateTime that is represented via a DatePicker control (available in .NET 4 only). On the managed code side, Visual Studio generates some starting code for querying data. In case you use the Entity Framework as the Data Access Layer, the generated code looks like this (continuing the example from the previous image):

```
Class MainWindow

    Private Function GetCategoriesQuery(ByVal NorthwindEntities As  _
                                        WpfApplication1.
NorthwindEntities) _
                                        As System.Data.Objects.
ObjectQuery(Of WpfApplication1.Category)

        Dim CategoriesQuery As System.Data.Objects.ObjectQuery(Of
WpfApplication1.Category) =
            NorthwindEntities.Categories
        'Update the query to include Products data in Categories. You
can modify this code as needed.
        CategoriesQuery = CategoriesQuery.Include("Products")
        'Returns an ObjectQuery.
        Return CategoriesQuery
    End Function

    Private Sub Window_Loaded(ByVal sender As System.Object,
                              ByVal e As System.Windows.
RoutedEventArgs) _
                              Handles MyBase.Loaded

        Dim NorthwindEntities As WpfApplication1.NorthwindEntities =
            New WpfApplication1.NorthwindEntities()
        'Load data into Categories. You can modify this code as
needed.
        Dim CategoriesViewSource As System.Windows.Data.
CollectionViewSource =
            CType(Me.FindResource("CategoriesViewSource"), System.
Windows.Data.CollectionViewSource)
        Dim CategoriesQuery As System.Data.Objects.ObjectQuery(Of
WpfApplication1.Category) =
            Me.GetCategoriesQuery(NorthwindEntities)
        CategoriesViewSource.Source = CategoriesQuery.
                                      Execute(System.Data.Objects.
MergeOption.AppendOnly)
    End Sub
End Class
```

This will allow loading some data when you run the application. Once you have your starting code, you can write your own queries or extend the auto-generated code with additional features, such as Insert/Update/Delete operations or provide data navigation functionalities.

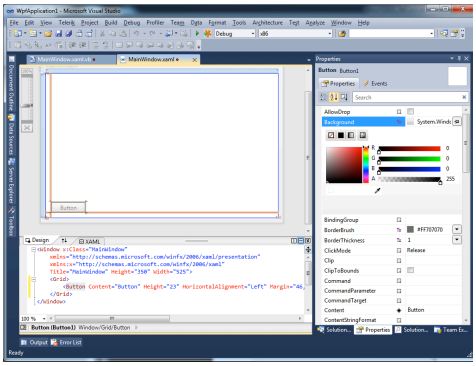
WPF & SILVERLIGHT: MULTI-TARGETING DESIGNER

In previous versions of Visual Studio the design-time experience for WPF and Silverlight developers did not offer specialized design tools. For WPF developers, the IDE lacked some tooling familiar to Windows Forms developers. Visual Studio 2008 introduced some improvements to the WPF design experience, like the ability to drag controls from the toolbox and drop them onto the designer surface, or the possibility to access event handlers by double-clicking the controls. But no support was available for setting special properties inside the Properties Window (e.g. brushes) so developers needed to manually write the code to set them. For Silverlight developers, things were even worse, in that you had neither drag and drop support nor design time support such as adjusting the application layout, so everything had to be set in code. Although Visual Studio is a developer tool, Microsoft Expression Blend is the tool best suited for deep customization of the user interface. While lots of developers use only Visual Studio to create the user interface for their applications, the need for better VS designer program was paramount. This is what happened with the release of Visual Studio 2010. In the next section you get an overview of the new and improved features, while later you will get information about how this targets multiple .NET Framework versions.

Improved experience

With Visual Studio 2010, both the WPF and Silverlight designers now provide a full design-time experience and, most importantly, both technologies share the same tools. This means that you have the same instrumentation available for both WPF and Silverlight projects. You now have full support for dragging and dropping controls from the toolbox, for rearranging controls on the controls' surface and you can also set complex properties within the Properties Window. The

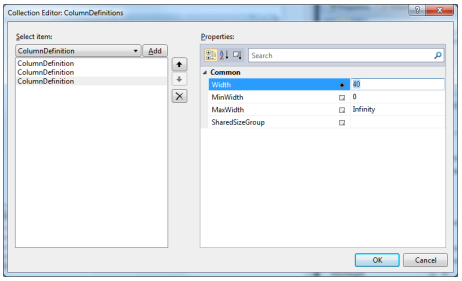
following figure represents such a description with regard to a WPF project:



If you try to do this inside a Silverlight project, you get exactly the same result and tools available. Notice how you now have a new color picker editor available for properties of type brush, which allows specifying the brush color, its type (e.g. a gradient or an image) and manually entering the color code. Prior to Visual Studio 2010 this had to be performed by writing pure XAML code. There are a number of other improvements in this area that are listed below.

Setting grid columns and rows

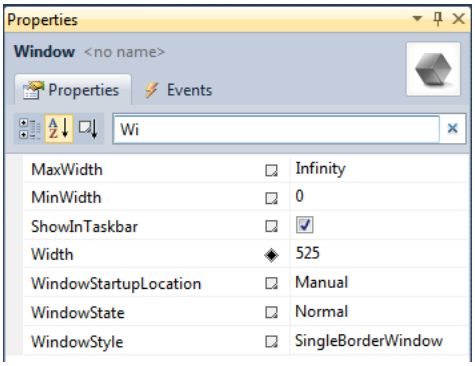
You can easily set rows and columns within a Grid control by selecting the RowDefinitions and ColumnDefinitions properties in the Property Window and then by taking advantage of a new dialog that looks like the following figure:



You simply click Add, then set properties for each row or column, such as the Height or Width.

Searching for a property

The Properties Window has been enhanced with a number of new features. You got a taste of this in the previous section when discussing how to set properties of type Brush. Another interesting addition is that you can now search for a specified property by writing its name in the search box at the top of the window. Also, you do not need to write the entire property name since Visual Studio filters the property list as you type characters inside the search box:



This way you can easily reach and set the properties you want to assign.

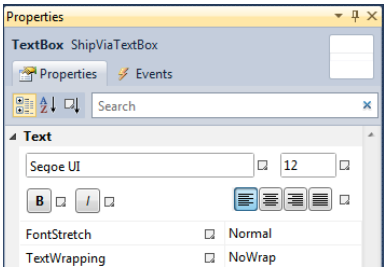
Hot Tip The search field is also available in the Events tab of the Properties Window.

Setting controls' fonts

In Visual Studio 2010 you can now easily set font properties to controls supporting fonts via an integrated tool inside the Properties Window.

Hot Tip This is available only when you enable the Categorized view.

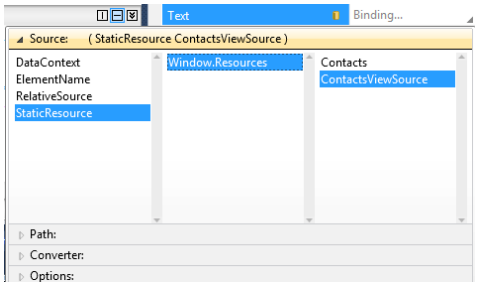
For instance, you can select the font from a combo box and then set text alignment, size and weight all together, as shown in the following figure:



Visually setting data-binding and resources

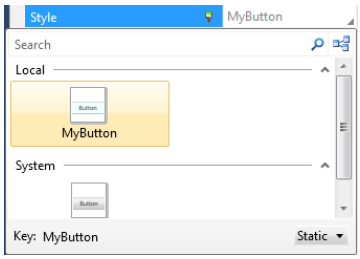
The Properties Window also improves the way you assign data-bound properties or resources. You select the data-bound property for your control and then you can easily use the Binding popup as follows:

1. Select your data source. This can be the DataContext—another control or a resource as shown in the following figure:



2. Switch to the Path tab in order to select the property you wish to be bound;
3. Select any value converter you want to apply via the Converter tab; and
4. Finally go to the Options tab and apply any other additional options. This is really impressive in that you have a great number of available properties such as data error validation or triggering property changes.

You can apply resources similarly. For example, if you defined a style for a button and then you want to apply such a style, just right click the Style property in the Properties Window and select Apply Resource. This will provide a useful popup where you will be able to pick both default and local resources, as demonstrated in the following figure:



You can also choose between static and dynamic resources and you are offered a preview of the result you get once the resource is applied.

Multiple .NET Framework versions targeted

Visual Studio 2010 ships with integrated support for Silverlight 3 but it also supports Silverlight 4 when you install the appropriate developer tools. The good news is that all the stuff you have seen so far, including the drag and drop data-binding, is available for both .NET Framework 3.5 SP 1 and 4.0 in Visual Studio 2010. This is what is usually referred to as multi-targeting, when talking about the designer. This basically means that you can create applications for WPF 3.5, WPF 4, Silverlight 3 and 4 taking advantage of all the new designer features independent of the technology or its version. If you want to check this out by yourself, simply create a new WPF or Silverlight project and in the New Project dialog select the .NET Framework 3.5 as the target Framework. You will notice no difference in the designer if compared to .NET 4.

COMMON SHORTCUTS AND TIPS

Visual Studio 2010 offers both shortcuts for quick access to common tools and instruments that you may find really useful. This section provides information on both topics.

Common shortcuts

- Start the application in debugging mode by pressing F5 or without attaching the debugger by pressing Shift+F5
- Save all files in your solution by pressing Ctrl+Shift+S
- Build your solution by pressing Ctrl+Shift+B
- Run your application without the debugger by pressing Shift+F5
- Show the toolbox by pressing Ctrl+Alt+T
- Navigate to a specific member definition by pressing Ctrl+
- Comment multiple line of codes by pressing CTRL+K and CTRL+C in sequence or uncomment lines by pressing CTRL+K and CTRL+U in sequence
- Show up the Properties window by pressing F4
- Select code blocks by pressing ALT + mouse
- Place a breakpoint by pressing F9 on the specified line of code

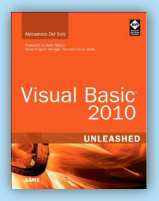
Common tips

- Download and manage Visual Studio extensions via the Tools|Extension Manager tool
- Add menu items for running external tools via the Tools|External Tools command
- Add IntelliSense code snippets within the code editor by pressing Ctrl+K or right-click and then Insert Snippet
- Export project or item templates in order to create reusable templates with File|Export template
- Find all references of a type or member by performing right click inside on a member inside the code editor and then selecting Find All References

ABOUT THE AUTHOR

Alessandro Del Sole, Microsoft Most Valuable Professional (MVP) for Visual Basic since 2008, is well known throughout the global Visual Basic community. The author of four books about .NET development with Visual Studio, he is a community leader on the Italian Visual Basic Tips and Tricks (www.visual-basic.it) web site that serves more than 42,500 VB developers. He is also a frequent contributor to the MSDN Visual Basic Developer Center (msdn.com/vbasic). He enjoys writing articles on .NET development, writing blog posts on both his Italian and English blogs, and producing instructional videos. You can find him online in forums or newsgroups.

RECOMMENDED BOOKS



This book's broad coverage includes advanced features such as generics and collections; a thorough introduction to the Visual Studio 2010 IDE and Visual Studio Team System; a full section on data access with ADO.NET and LINQ; practical overviews of WPF and WCF; coverage of web and cloud development with Silverlight and Azure; and advanced topics such as multithreading, testing, and deployment.

DZone communities deliver over 6 million pages each month to more than 3.3 million software developers, architects and decision makers. DZone offers something for everyone, including news, tutorials, cheat sheets, blogs, feature articles, source code and more. **"DZone is a developer's dream,"** says PC Magazine.

DZone, Inc.
140 Preston Executive Dr.
Suite 100
Cary, NC 27513

888.678.0399
919.678.0300

Refcardz Feedback Welcome
refcardz@dzone.com

Sponsorship Opportunities
sales@dzone.com

ISBN-13: 978-1-934238-75-2
ISBN-10: 1-934238-75-9
50795
9 781934 238752
$7.95

Version 1.0