Click to verify

Data binding: The DataContext property is the default source of your bindings, unless you specifically declare another source, like we did in the previous chapter with the ElementName property. It's defined on the FrameworkElement class, which most UI controls, including the WPF Window, inherits from. Simply put, it allows you to specify a basis for your bindingsThere's no default source for the DataContext property (it's simply null from the start), but since a DataContext is inherited down through the control hierarchy, you can set a DataContext for the Window itself and then use it throughout all of the child controls. Let's try illustrating that with a simple example: using System;using System.Windows;namespace WpfTutorialSamples.DataBinding{public partial class DataContextSample : Window{public DataContextSample(){InitializeComponent();this.DataContext = this;}}}The Code-behind for this example only adds one line of interesting code: After the standard InitalizeComponent() call, we assign the "this" reference to the DataContext, which basically just tells the Window that we want itself to be the data context. In the XAML, we use this fact to bind to several of the Window properties, including Title, Width and Height. Since the window has a DataContext, which is passed down to the child controls, we don't have to define a source on each of the bindings - we just use the values as if they were globally available.Try running the example and resize the window - you will see that the dimension changes are immediately reflected in the textboxes. You can also try writing a different title in the first textbox, but you might be surprised to see that this change is not reflected immediately. Instead, you have to move the focus to another control before the change is applied. Why? Well, that's the subject for the next chapter.SummaryUsing the DataContext property is like setting the basis of all bindings down through the hierarchy of controls. This saves you the hassle of manually defining a source for each binding, and once you really start using data bindings, you will definitely appreciate the time and typing saved.However, this doesn't mean that you have to use the same DataContext for all controls within a Window. Since each control has its own DataContext property, you can easily break the chain of inheritance and override the DataContext with a new value. This allows you to do stuff like having a global DataContext on the window and then a more local and specific DataContext on e.g. a panel holding a separate form or something along those lines. This article has been fully translated into the following languages: Is your preferred language not on the list? Click here to help us translate this article into your language! I have a TextBlock that is bound to a DateTime property. How do I configure the format of the date? There is a string format property available when you are declaring the binding: (You need to be on .NET 3.5 SP1 for this property to exist) If you want to use a common format string between bindings, you could declare the binding like this: With your constants class like this: public static class Constants{ public const string DateTimeUiFormat = "dd/MM/yyyy"; //etc...} May be helpful to someone: or 24h and 2digits month and year format: In an usual application sometimes you need to adapt the values from the view model. This is normally done using StringFormat, but well see some other options as well. Simple StringFormat with binding escapeLets say that you need to display a temperature in degrees. In the view model you have just the numerical value and in the interface you want to append the C string to make it clear what type of degrees are displayed. Heres how thats done: MultiBindingThe zero from XAML binding is actually the first binding. In the next example Name is the {0} part and ID is the {1} part: There are however some other ways you can concatenate string values in XAML. Lets review them quickly: TextBlock with Run text Using StackPanel to group Using Converters TextBlock with Run textTextBlock supports an inner element called Run which can be helpful when you want to concatenate more things. Using StackPanel to groupIn this case you can just dump everything in a StackPanel having the Orientation set to Horizontal. Using ConverterAnd the last example, although I wouldnt really use it in this case (but shown nonetheless just for completeness sake): public class TemperatureConverter : IValueConverter { public object Convert(object value, Type targetType, object parameter, CultureInfo culture) { return $"Temperature is {value} C"; } public object ConvertBack(object value, Type targetType, object parameter, CultureInfo culture) { throw new NotImplementedException(); }} And heres how to use it in XAML: Most common formatting specifiersFormatting numbers using 2 decimal points is done using F2 - F means floating point and the following digit is the number of decimal digits. In this case I used 2, but it can be any value. If you want to show only the integral part then use F0. If you also want to display the thousands separator you can use N2. Here are some more examples showing how to display currency and dates: Resources It is often desirable to control the format of dates, times, money and percentages when they are displayed. The example screen-shot below shows a list of 3 sale records. The sale information includes an Id, the sale amount, the percentage of sales tax paid and the time of the sale. Although the information displayed to the user is correct, the unformatted data might be difficult or confusing for a user to read. The .NET framework includes a small formatting mini-language using strings like "{0:C}" to format a parameter as currency. WPF data binding does not have any facility built-in to allow this formatting, however it is easy to add by extending the built-in binding infrastructure. This technique I originally came across here in the WPF forums, but it has been extended here to allow formatting of all data types. Implementing a custom ValueConverter for FormattingTo extend the binding infrastructure to allow formatting I've created a ValueConverter to convert between a .NET object and a string. Value converters implement the IValueConverter interface in the System.Windows.Data namespace. This interface has two methods, Convert and ConvertBack for converting between two types. To convert the data from a .NET object to a formatted string we're only going to implement the Convert() method. The formatting string (like "{0:MM-dd-yyyy}" if we wished to format a date) is passed to the Convert() method of the ValueConverter via the aptly named parameter parameter. Fortunately the Convert() method is also passed a CultureInfo parameter which we can use to perform our formatting in a culture-specific way. Formatting should be culture aware, as different countries use different symbols for currency, have different date time formats and different symbols to separate the whole and fractional parts of non-integer real numbers. The code for the ValueConverter is shown below: C# Code (Feb 2006 CTP) using System;using System.Windows.Data; namespace LearnWPF.FormattingAndDataBinding{ [ValueConversion(typeof(object),typeof(string))] public class FormattingConverter: IValueConverter { public object Convert(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture) { string formatString = parameter as string; if (formatString != null) { return string.Format(culture, formatString, value); } else { return value.ToString(); } } public object ConvertBack(object value, Type targetType, object parameter, System.Globalization.CultureInfo culture) { // we don't intend this to ever be called return null; } }} Using the Converter from our WPF ApplicationTo use the converter in our WPF application requires 3 steps: 1 - Create an Xml namespace prefix for the CLR namespace that contains the ValueConverter. This is necessary any time you want to use one of your own types in mark-up. The example below tells the Xaml run-time that any time it sees a "my" prefix on an element it should look in the LearnWPF.FormattingAndDataBinding namespace in the current assembly. More details of this can be found here in the Windows SDK. xmlns:my="clr-namespace:LearnWPF.FormattingAndDataBinding" 2 - Create an instance of our formatter as a resource*. In this sample I've chosen to place it in the resources dictionary for the window. A key is required so we can retrieve it later. 3 - Use the Converter when binding. We do this by specifying a Converter for the binding (the converter we created in step 2 as a resource), and also setting the ConverterParameter. The ConverterParameter is passed to the IValuConverter's Convert() method as the (aptly named) parameter parameter. The example below passes a custom date format string {0:dd-MMM-yyy hh:mm}. Note that in the format string the curly brackets are escaped using the backslash character like this \{ and this \}. This is necessary because the curly brackets are used in Xaml for specifying markup extensions like that {Binding} and {StaticResource} extensions seen in this example. We need to differentiate our formatting string from those. Here is the Xaml code containing all three steps Basic controls: The CheckBox control allows the end-user to toggle an option on or off, usually reflecting a Boolean value in the Code-behind. Let's jump straight into an example, in case you're not sure how a CheckBox looks: Application OptionsEnable feature ABCEnable feature XYZEnable feature WWW As you can see, the CheckBox is very easy to use. On the second CheckBox, I use the IsChecked property to have it checked by default, but other than that, no properties are needed to use it. The IsChecked property should also be used from Code-behind if you want to check whether a certain CheckBox is checked or not.Custom contentThe CheckBox control inherits from the ContentControl class, which means that it can take custom content and display next to it. If you just specify a piece of text, like I did in the example above, WPF will put it inside a TextBlock control and display it, but this is just a shortcut to make things easier for you. You can use any type of control inside of it, as we'll see in the next example: Application OptionsEnable feature ABCEnable feature XYZEnable feature WWWAs you can see from the sample markup, you can do pretty much whatever you want with the content. On all three check boxes, I do something differently with the text, and on the middle one I even throw in an Image control. By specifying a control as the content, instead of just text, we get much more control of the appearance, and the cool thing is that no matter which part of the content you click on, it will activate the CheckBox and toggle it on or off.The IsThreeState propertyAs mentioned, the CheckBox usually corresponds to a boolean value, which means that it only has two states: true or false (on or off). However, since a boolean data type might be nullable, effectively allowing for a third option (true, false or null), the CheckBox control can also support this case. By setting the IsThreeState property to true, the CheckBox will get a third state called "the indeterminate state".A common usage for this is to have a "Enable all" CheckBox, which can control a set of child checkboxes, as well as show their collective state. Our example shows how you may create a list of features that can be toggled on and off, with a common "Enable all" CheckBox in the top:Application OptionsEnable allEnable feature ABCEnable feature XYZEnable feature WWWusing System;using System.Windows;namespace WpfTutorialSamples.Basic_controls{public partial class CheckBoxThreeStateSample : Window{public CheckBoxThreeStateSample(){InitializeComponent();}private void cbAllFeatures_CheckedChanged(object sender, RoutedEventArgs e){bool newVal = (cbAllFeatures.IsChecked == true);cbFeatureAbc.IsChecked = newVal;cbFeatureXyz.IsChecked = newVal;cbFeatureWww.IsChecked = newVal;}private void cbFeature_CheckedChanged(object sender, RoutedEventArgs e){cbAllFeatures.IsChecked = null;if((cbFeatureAbc.IsChecked == true) && (cbFeatureXyz.IsChecked == true) && (cbFeatureWww.IsChecked == true))cbAllFeatures.IsChecked = true;if((cbFeatureAbc.IsChecked == false) && (cbFeatureXyz.IsChecked == false) && (cbFeatureWww.IsChecked == false))cbAllFeatures.IsChecked = false;}}This example works from two different angles: If you check or uncheck the "Enable all" CheckBox, then all of the child check boxes, each representing an application feature in our example, is either checked or unchecked. It also works the other way around though, where checking or unchecking a child CheckBox affects the "Enable all" CheckBox state: If they are all checked or unchecked, then the "Enable all" CheckBox gets the same state - otherwise the value will be left with a null, which forces the CheckBox into the indeterminate state.All of this behavior can be seen on the screenshots above, and is achieved by subscribing to the Checked and Unchecked events of the CheckBox controls. In a real world example, you would likely bind the values instead, but this example shows the basics of using the IsThreeState property to create a "Toggle all" effect. This article has been fully translated into the following languages: Is your preferred language not on the list? Click here to help us translate this article into your language! Data binding: As we saw in the previous chapters, the way to manipulate the output of a binding before it is shown is typically through the use of a converter. The cool thing about the converters is that they allow you to convert any data type into a completely different data type. However, for more simple usage scenarios, where you just want to change the way a certain value is shown and not necessarily convert it into a different type, the StringFormat property might very well be enough.Using the StringFormat property of a binding, you lose some of the flexibility you get when using a converter, but in return, it's much simpler to use and doesn't involve the creation of a new class in a new file. The StringFormat property does exactly what the name implies: It formats the output string, simply by calling the String.Format method. Sometimes an example says more than a thousand words, so before I hit that word count, let's jump straight into an example:The first couple of TextBlock's gets their value by binding to the parent Window and getting its width and height. Through the StringFormat property, the values are formatted. For the width, we specify a custom formatting string and for the height, we ask it to use the currency format, just for fun. The value is saved as a double type, so we can use all the same format specifiers as if we had called double.ToString(). You can find a list of them here: Also notice how I can include custom text in the StringFormat - this allows you to pre/post-fix the bound value with text as you please. When referencing the actual value inside the format string, we surround it by a set of curly braces, which includes two values: A reference to the value we want to format (value number 0, which is the first possible value) and the format string, separated by a colon.For the last two values, we simply bind to the current date (DateTime.Now) and the output it first as a date, in a specific format, and then as the time (hours and minutes), again using our own, pre-defined format. You can read more about DateTime formatting here: without extra textPlease be aware that if you specify a format string that doesn't include any custom text, which all of the examples above does, then you need to add an extra set of curly braces, when defining it in XAML. The reason is that WPF may otherwise confuse the syntax with the one used for Markup Extensions. Here's an example:Using a specific CultureIf you need to output a bound value in accordance with a specific culture, that's no problem. The Binding will use the language specified for the parent element, or you can specify it directly for the binding, using the ConverterCulture property. Here's an example:It's pretty simple: By combining the StringFormat property, which uses the D specifier (Long date pattern) and the ConverterCulture property, we can output the bound values in accordance with a specific culture. Pretty nifty! This article has been fully translated into the following languages: Is your preferred language not on the list? Click here to help us translate this article into your language! So you want to format the output of information but dont want do it in code behind or write a value converter to do it. Well, good news, you dont have to. You can format your data directly in XAML. How, you may be asking? New in .NET 3.5 SP1 is the StringFormat attribute. Example 1: Lets say you want to format a double value into a currency: Notice the { } just after the StringFormat attribute? What that is doing is escaping the text after the = sign. You need to do this because we do not have any text directly after the = sign. If you dont put the { }, strange things will happen. And now lets say we want to place some text in front of the currency: Since we now have text after the = sign we no longer need the { }. How about a date you ask: Oh, and you want time: What? You want to create a tooltip comprised of more than one property of an object. Well Okay: As you can see the StringFormat attribute can be a time saver, and just make life a little easier. One thing to note is that if you use the StringFormat attribute and you bind to a property that has no value, otherwise known as null, then the text that will be displayed is {DependencyProperty.UnsetValue}". Data binding: Wikipedia describes the concept of data binding very well:Data binding is general technique that binds two data/information sources together and maintains synchronization of data. With WPF, Microsoft has put data binding in the front seat and once you start learning WPF, you will realize that it's an important aspect of pretty much everything you do. If you come from the world of WinForms, then the huge focus on data binding might scare you a bit, but once you get used to it, you will likely come to love it, as it makes a lot of things cleaner and easier to maintain.Data binding in WPF is the preferred way to bring data from your code to the UI layer. Sure, you can set properties on a control manually or you can populate a ListBox by adding items to it from a loop, but the cleanest and purest WPF way is to add a binding between the source and the destination UI element.SummaryIn the next chapter, we'll look into a simple example where data binding is used and after that, we'll talk some more about all the possibilities. The concept of data binding is included pretty early in this tutorial, because it's such an integral part of using WPF, which you will see once you explore the rest of the chapters, where it's used almost all of the time. However, the more theoretical part of data binding might be too heavy if you just want to get started building a simple WPF application. In that case I suggest that you have a look at the "Hello, bound world!" article to get a glimpse of how data binding works, and then save the rest of the data binding articles for later, when you're ready to get some more theory. wpf-tutorial.com 2007-2025 Contact Us Suggest Correction Localization Data binding: StringFormat StringFormat StringFormat String.Format StringFormat doubledouble.ToString() StringFormat (DateTime.Now) DateTime Window x:Class="WpfTutorialSamples.DataBinding.StringFormatSample" xmlns=" xmlns:x=" xmlns:system="clr-namespace:System;assembly=mscorlib" Title="StringFormatSample" Height="150" Width="250"Name="wnd">ConverterCulture D StringFormat ConverterCulture This article has been fully translated into the following languages: Is your preferred language not on the list? Click here to help us translate this article into your language! Data binding: As we saw in the previous data binding examples, defining a binding by using XAML is very easy, but for certain cases, you may want to do it from Code-behind instead. This is pretty easy as well and offers the exact same possibilities as when you're using XAML. Let's try the "Hello, bound world" example, but this time create the required binding from Code-behind: using System;using System.Windows;using System.Windows.Controls;using System.Windows.Data;namespace WpfTutorialSamples.DataBinding{ public partial class CodeBehindBindingsSample : Window { public CodeBehindBindingsSample() { InitializeComponent(); Binding binding = new Binding("Text"); binding.Source = txtValue; lblValue.SetBinding(TextBlock.TextProperty, binding); } }}It works by creating a Binding instance. We specify the path we want directly in the constructor, in this case "Text", since we want to bind to the Text property. We then specify a Source, which for this example should be the TextBox control. Now WPF knows that it should use the TextBox as the source control, and that we're specifically looking for the value contained in its Text property.In the last line, we use the SetBinding method to combine our newly created Binding object with the destination/target control, in this case the TextBlock (lblValue). The SetBinding() method takes two parameters, one that tells which dependency property that we want to bind to, and one that holds the binding object that we wish to use. SummaryAs you can see, creating bindings in C# code is easy, and perhaps a bit easier to grasp for people new to data bindings, when compared to the syntax used for creating them inline in XAML. Which method you use is up to you though - they both work just fine. This article has been fully translated into the following languages: Is your preferred language not on the list? Click here to help us translate this article into your language! You signed in with another tab or window. Reload to refresh your session. You signed out in another tab or window. Reload to refresh your session. You switched accounts on another tab or window. Reload to refresh your session. Dismiss alert Instantly share code, notes, and snippets. Clone this repository at <script src=" quot;></script> Save ebersys/5144657 to your computer and use it in GitHub Desktop. Clone this repository at <script src=" quot;></script> Save ebersys/5144657 to your computer and use it in GitHub Desktop. WPF: Bind formatted DateTime, using a constant if you want to reuse, or actual StringFormat You cant perform that action at this time.

Wpf textblock datetime format. Wpf textblock stringformat date. Wpf textblock stringformat datetime.