DOWNLOAD PDF CUSTOM CONTROLS LIBRARY

Chapter 1 : Custom controls library (Book,) [calendrierdelascience.com]

In practice custom controls are something you implement on the code level while you can use XAML for user controls. The custom controls extend one of the WPF control base classes and provide additional functionality through code so all the added logic and representation must be implemented inside the code.

How a user control is processed When a page with a user control is requested, the following occurs: The page parser parses the. The parser then dynamically compiles the class into an assembly. If you are using Visual Studio, then at design time only, Visual Studio creates a code behind file for the user control, and the file is precompiled by the designer itself. Finally, the class for the user control, which is generated through the process of dynamic code generation and compilation, includes the code for the code behind file. What are custom controls? Custom controls are compiled code components that execute on the server, expose the object model, and render markup text, such as HTML or XML, as a normal Web Form or user control does. How to choose the base class for your custom control To write a custom control, you should directly or indirectly derive the new class from the System. Control class or from the System. You should derive from System. Control if you want the control to render nonvisual elements. WebControl if you want the control to render HTML that generates a visual interface on the client computer. If you want to change the functionality of existing controls, such as a button or label, you can directly derive the new class with these existing classes and can change their default behavior. In brief, the Control class provides the basic functionality by which you can place it in the control tree for a Page class. The WebControl class adds the functionality to the base Control class for displaying visual content on the client computer. For example, you can use the WebControl class to control the look and styles through properties like font, color, and height. How to create and use a simple custom control that extends from System. Create a class library project, and give it a name, for example, CustomServerControlsLib. Add a source file to the project, for example, SimpleServerControl. Include the reference of the System. Web namespace in the references section. Check whether the following namespaces are included in the SimpleServerControl.

Chapter 2 : Creating Custom WPF Controls

Process Instruments Wireless Measurement Wireless Valve Diagnostics SCADA RTUs Radio Systems Sensors & Barriers Industrial Automation Variable Speed Drives Moisture Analyzers Industrial Video And Control Calibration Equipment Compressor Controls Indicators SCADA Software Tank Gauging Flow Meters.

But the challenge is to do it the right way. So before you start creating a control try to answer the following questions: What problem should my control solve? Who will use this control? In which context and environment? Can I extend or compose existing controls? Have a look at Existing Controls? Should it be possible to style or template my control? What design-time support should it have? In Expression Blend and Visual Studio? Is it used in a single project, or part of a reusable library? This is the place where our custom control comes in. This is place where we test our control in a simple application. Add a reference to the custom control library by using the "Add Reference" context menu entry on the "PopupControlTest" project item in the solution explorer. Rename the CustomControl1 to PopupControl. Choose the right base class Choosing the right base class is crucial and can save a lot of time! Compare the features of your control with existing controls and start with one that matches close. The following list should give you a good overview from the most leightweight to more heavyweight base types: UIElement - The most lightweight base class to start from. It is first base class that takes part in the logical tree and so it supports data binding and resource lookup. Control - is the most common base class for controls its name speaks for itself. It supports templates and adds some basic properties as Foreground, Background or FontSize. ContentControl - is a control that has an additional Content property. This is often used for simple containers. HeaderedContentControl - is a control that has an Content and a Header property. ItemsControl - a control that has an additional Items collection. This is a good choice for controls that display a dynamic list of items without selection. Selector - an ItemsControl whose items can be indexed and selected. RangeBase - is the base class for controls that display a value range like Sliders or ProgressBars. It adds an Value, Minimum and Maximum property. The behavior is defined in code. The template is defined in XAML. The default template is by convention wrapped into a style that has an implicit key. That is is not a string - as usually - but a Type object of our control. And that is excactly what we are doing in the static constructor. We are overriding the default value of the DefaultStyleKey property and set it to the Type object of our control. Create a default Style The style must be located by convention in a folder called "Themes" that must be located in the root of the control library project. In these folder we can provide different templates for each Windows theme. The name of these ResourceDictionaries must match the name of the windows theme. If we do not provide any theme-specific styles, we need to provide the fallback style located in the "Generic. As we set the default value of the DefaultStylekey property to the Type object of our control, we must give our default style the same key to be found. This is done by leaving the x: Map XML Namespace 7. Overriding GetLogicalChildren creates the connection to navigate down the logical tree tunneling.

Chapter 3 : Library of Custom Controls - Discussion Forums - National Instruments

Custom Controls Library. A set of miscellaneous WPF custom controls. Busy Indicator. A simple control to indicate that the application is busy. calendrierdelascience.com calendrierdelascience.com are.

Inheriting from the WebControl Class If your control renders a user interface UI element or any other visible element on the client, you should derive your control from System. WebControl or a from class that derives from it. In this example the custom control derives from Label , which in turn derives from System. If your control renders an element that is not visible in the client browser, such as a hidden element or a meta element, derive your control from System. NET without any extra work on your part. On each postback, the page is re-created and values are restored from view state. If the DefaultUserName property value were not stored in view state, the value would be set to its default, Empty , on each postback. The ViewState property, which is inherited from WebControl , is a dictionary that saves data values. Values are entered and retrieved using a String key. In this case, "DefaultUserName" is used as the key. Items in the dictionary are typed as Object , and when you access them, you must cast them to the actual property type. For more information about view state, see ASP. This improves performance because the HtmlTextWriter object writes directly to the output stream. String concatenation requires time and memory to create the string, and then writes to the stream. In general, when your control derives from WebControl and renders a single element, you should override the RenderContents method and not the Render method. The Render method of WebControl invokes RenderContents after rendering the opening tag for the control and its style attributes. If you override the Render method to write contents, your control will lose the style-rendering logic that is built into the Render method of WebControl. Attributes of the Control The attributes that are applied to the WelcomeLabel control and to the DefaultUserName property contain metadata that is used by the common language runtime and by design-time tools. At the class level, WelcomeLabel is marked with the following attributes: This is a design-time attribute that specifies the default property of a control. In visual designers, the property browser typically highlights the default property when a page developer clicks the control on the design surface. This specifies the format string for the element. For WelcomeLabel, the string creates the following element: These attributes are applied as ParseChildren true and PersistChildren false. The attributes work together and with the ToolboxDataAttribute attribute so that child elements are interpreted as properties and properties are persisted as attributes. Attributes of the DefaultUserName Property The following attributes that are applied to the DefaultUserName property of WelcomeLabel are design-time attributes that you typically apply to all public properties of your controls: This specifies for visual designers whether it is meaningful to bind the property to data. For example, in Visual Studio, if a property is marked with Bindable true , the property is displayed in the DataBindings dialog box. If a property is not marked with this attribute, the property browser infers the value to be Bindable false. For example, Category "Appearance" tells the property browser to display the property in the Appearance category when the page developer uses the category view of the property browser. You can specify a string argument that corresponds to an existing category in the property browser, or you can create your own category. This specifies a brief description of the property. In Visual Studio, the property browser displays the description of the selected property at the bottom of the Properties window. This specifies a default value for the property. This value should be the same as the default value you return from the property accessor getter. In Visual Studio, the DefaultValueAttribute attribute enables a page developer to reset a property value to its default by displaying the shortcut menu in the Properties window and clicking the Reset button. This specifies for visual designers whether it is meaningful to localize the property. When a property is marked Localizable true , the visual designer includes the property value when the property is serialized as a resource. The designer will persist the property value to the culture-neutral resource file or to another localization source when the control is polled for localizable properties. Design-time attributes that are applied to a control and to its members do not affect how the control functions at run time. However, they enhance the developer experience when the control is used in a visual designer. For a complete listing of design-time and run-time attributes for server controls, see Metadata Attributes for Custom Server

Controls. This is useful if the custom control will be used in multiple pages in a Web application. The following example shows a Web.

DOWNLOAD PDF CUSTOM CONTROLS LIBRARY

Chapter 4 : Controls | Microsoft Docs

In moving an app from Access to a C#/SQL Server Windows app, I'd like to make some custom controls, by I am having trouble finding a discussion of how to do this.

Please Sign up or sign in to vote. Reusability and factorizing are maybe the most commons things you want and use when you are developing applications. The example to illustrate the theory will be to create an headered control. Creating a control library Foundation of the project The first step is to use the VS Wizard named "WPF Custom control library" to create the library which will contain the controls. We will name it, by excess of modesty: As you can see, VS has created some files and directory for us: Here is its content: It tells the framework where to look for the theme files. Usually it is used like this: Its content is interesting because it show you that you have to override the MetaData of the newly created control to force the WPF engine using the style defined in the generic. Of course, the wizard to add a new Custom Control can do it for you but its always a good thing to know how it works. Here we are with the foundation of the library. Now we are going to add a template and behaviors to the control. Choose the more appropriated base class By default the wizard make the control a inherit from the Control class but there are differents options that you may consider for base class: This is a base class which lets you the most freedom but the most to do! This is a good choice if the control will be used to represent a single piece of content. This is a good choice to represent a single element with an header. This is a good choice if the control will be used to represent a list of items. Note that it meets the specifications: Adding a Template and some behaviors to the control Setting the template of the control Unlike UserControl there is no code-behind and you set the template of the control by creating a style in the generic. By revelants parts, I mean the elements which together build the behavior of your control: The content can be of any type of your choice and is usually placed in the Content property for ContentControl and the Items property for ItemsControl. We also use a TemplateBinding to bind the content of the class to the ContentPresenter. In our case we add the content under the header: This is something very common that you use often without noticing it: In our case there is a lot which are already here because of our inheritance of FrameworkElement. In our case we add two property: So we have this class: Dock value only the revelant XAML is showed: Create the RoutedCommand of your choice in the control class, Create a private method which will be called when the command is executed, Register a Class Command binding of this command in the static constructor of the control you can think of it as a static command binding which will be called each time the command is executed , In the handler of the class command binding gets the sender of the command, cast it to your class and call the private method you created just before. Adding a Routed event to the control Adding an event can be useful to notify the others part of your application that an action occurred in the control or to deliver informations. To add an event here are the guideline: Now lets assume we wants to raise an event when the header is clicked. Simply that when you will use the controls in another page you will not declare the XML namespace via an assembly name but via an URL easy to remember.

DOWNLOAD PDF CUSTOM CONTROLS LIBRARY

Chapter 5 : Writing your Custom Control: step by step - CodeProject

*Custom Controls Library [Rod Stephens] on calendrierdelascience.com *FREE* shipping on qualifying offers. For programmers who don't know Visual Basic or have CCE, this book allows them to take advantage of this technology.*

Follow Despite the enormous quantity of controls supplied by SAPUI5 framework it is likely that at one or another moment you realise that your end users go whimsical and you face a task of creating another Control successor. Not a big deal actually, just open a corresponding guide and off you go. You implement a new control and use it as a part of an App project. However, sooner or later you will need to use the same control in another project, then in one more, then in yet another, and so forth. And at one moment I found myself in such a situation. After quite an hours of practicing trial-and-error technique I found a solution: Unfortunately I cannot guarantee that the technique described below would be applicable to any installations, but hey, we are all developers, and the more challenging task is the more fun to resolve it. Implementing Library A brief review of the necessary steps of library implementation can be found here. To start implementing library in the Web IDE first you have to create a project. We also assume that the library will contain a single control ProductRating taken from the OpenUI5 tutorial. Each subfolder should contain library. And here the first problem arises: And further work showed it was a correct first step. In the dialog shown next you will be presented with default values including application public name copy it for further use and version as in the following figure: Yes, you enter all the necessary dependencies into your JS controller files and appropriate name spaces into XML views, however it appears that Web IDE still does not know of existence of some shareable application where you store your library files. To resolve this, we need to amend another configuration file found in the project root folder â€" neo-app. Normally for a new porject the file neo-app. In our case we need to tell a similar thing about our library. When mentioning our ProductRating control from the new library in some controller file the dependency string should be like this: After that you can run the App and it should correctly load resources from the deployed library. Remember though that once you have changed something in the library files you have to redeploy library again. Next time you deploy the library the Web IDE will suggest the new version of the existing HCP application, so most probably you accept all the default values. There is no need to update App porject using the library as by default Web IDE will load resources from the most recent version of the library.

Chapter 6 : Custom Controls | Microsoft Docs

custom controls ready to plug into any program. This book contains instructions for installing and using the controls with Visual Basic, Visual C++, C++ Builder, VBScript, JavaScript, Visual J++, Java, and Delphi.

Historically, these objects have been referred to as controls. While the WPF SDK continues to use the term "control" to loosely mean any class that represents a visible object in an application, it is important to note that a class does not need to inherit from the Control class to have a visible presence. This topic discusses how controls both those that do inherit from the Control class and those that do not are commonly used in WPF. The following example shows how to create a simple application that asks a user for their first and last name. This example creates six controls: All controls can be created similarly. For brevity, the creation of the Grid , grid1, has been excluded from the sample. SetColumn firstName, 1 ; grid1. SetRow lastNameLabel, 1 ; grid1. SetColumn lastName, 1 ; Grid. SetRow lastName, 1 ; grid1. SetRow submit, 2 ; grid1. SetRow clear, 2 ; Grid. SetColumn clear, 1 ; grid1. SetColumn firstName, 1 grid1. SetRow lastNameLabel, 1 grid1. SetColumn lastName, 1 Grid. SetRow lastName, 1 grid1. SetRow submit, 2 grid1. SetRow clear, 2 Grid. SetColumn clear, 1 grid1. You can change the appearance of a control by doing one of the following, depending on what you want to accomplish: Change the value of a property of the control. Create a Style for the control. Create a new ControlTemplate for the control. You can set the value properties in both XAML and code. Add new GradientStop Colors. Green, 0 ; buttonBrush. Add New GradientStop Colors. Bold Creating a Style for a Control WPF gives you the ability to specify the appearance of controls wholesale, instead of setting properties on each instance in the application, by creating a Style. The following example creates a Style that is applied to each Button in the application. For more information about styles, see Styling and Templating. Creating a ControlTemplate A Style allows you to set properties on multiple controls at a time, but sometimes you might want to customize the appearance of a Control beyond what you can do by creating a Style. Classes that inherit from the Control class have a ControlTemplate , which defines the structure and appearance of a Control. The Template property of a Control is public, so you can give a Control a ControlTemplate that is different than its default. You can often specify a new ControlTemplate for a Control instead of inheriting from a control to customize the appearance of a Control. Consider the very common control, Button. The primary behavior of a Button is to enable an application to take some action when the user clicks it. By default, the Button in WPF appears as a raised rectangle. In this case, you can create a new ControlTemplate. The following example creates a ControlTemplate for a Button. The ControlTemplate creates a Button with rounded corners and a gradient background. When you set the Background property of the Button , the color of that value will be used as the first GradientStop. For more information about data binding, see Data Binding Overview. The example also creates a Trigger that changes the appearance of the Button when IsPressed is true. The following example shows how to subscribe to the Click event of a Button. For example, a Label can contain any object, such as a string, an Image , or a Panel. The following classes provide support for rich content and act as base classes for most of the controls in WPF.

Chapter 7 : How to Create Windows Control Library and how to use in C#.Net

A thread explains to choose File --> New Project --> Windows Type --> WPF Custom control library template. But I don't find any such template in VS Found a web link explaining how to create a VS template for custom library.

Chapter 8 : Developing Custom calendrierdelascience.com Server Controls

This section contains information about application-defined or custom controls. The following topics are discussed. Buttons, menus, static text controls, list boxes, and combo boxes can be created with an owner-drawn style flag. When a control has the owner-drawn style, the system handles the user's.

Chapter 9 : Custom Controls Library

Now in the library file register the custom controls under control section. I have kept the library file as simple as possible. Also the library namespace is same as in manifest i.e. "calendrierdelascience.coml".