PETER'S TEXTBOXES USER'S GUIDE

peterblum.com
Start with better controls. Finish with better sites.

Click on any of these topics to jump to them:

- Enhanced TextBox Control ..... Using Adding Properties
- IntegerTextBox Control ..... Using Adding Properties
- DecimalTextBox Control ..... Using Adding Properties
- CurrencyTextBox Control ..... Using Adding Properties
- PercentTextBox Control ..... Using Adding Properties
- FilteredTextBox Control ..... Using Adding Properties
- MultiSegmentDataEntry Control ..... Using Adding Properties
  - Segments of the MultiSegmentDataEntry Control ..... Using Adding Properties
- Page Level Properties (The PeterBlum.DES.Globals.Page object)
- Validation with the Native Validation Framework
- JavaScript Support Functions
- Troubleshooting
- Table Of Contents

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Table of Contents

License Information 7

Platform Support 7

Technical Support and Other Assistance 8
- Troubleshooting Section of this Guide 8
- Developer's Kit 8
- PeterBlum.Com MessageBoard 8
- Getting Product Updates 8
- Technical Support 8

WHAT DOES PETER'S DATA ENTRY SUITE DO? 10

PETER'S TEXTBOXES OVERVIEW 13

Enhanced TextBox Overview 13

IntegerTextBox, DecimalTextBox, CurrencyTextBox, and PercentTextBox Overview 14

FilteredTextBox Overview 14

MultiSegmentDataEntry Control Overview 15

Date and Time Entry TextBoxes 16

Other Data Entry Controls 16

ENHANCED TEXTBOX CONTROL 17

Features 18

Using the Enhanced TextBox Control 19
- Getting and Setting the Value of the TextBox 19
  - Data Entry Validation 19
- When the TextBox is Empty 21
- Validation on AutoPostBack 22
- Interactive Hints 23
  - Setting Up Hints with PopupViews 24
  - Setting Up Hints in a Label or Panel 25
- AutoComplete and "Smart Change System" 26
- Other Behaviors 27

Adding an Enhanced TextBox 28

Converting the ASP.NET TextBox to the Enhanced TextBox 30

Properties for the Enhanced TextBox 31
- Getting And Setting the Value Properties 31
- Editing Properties 32
- Behavior Properties 34
- Appearance Properties 35
- AutoPostBack Properties 36
- Value When Blank Properties 37

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

ToolTip Properties 38
Hint Properties 39
Tab Rules Properties 41
Client-Side Functions Properties 43

INTEGERTEXTBOX CONTROL 45

Features 46

Using the IntegerTextBox 47
Getting and Setting the Value of the TextBox 48
Data Entry Validation 48
Formatting The Text 49
Adding A Spinner 50
Connecting Data To Other Fields On The Client-Side 51

Adding a IntegerTextBox 52

Properties for the IntegerTextBox 54
Getting And Setting The Value Properties 55
Editing Properties 57
Formatting Properties 59
Spinner Properties 60

DECIMALTEXTBOX CONTROL 62

Features 63

Using the DecimalTextBox 64
Getting and Setting the Value of the TextBox 65
Data Entry Validation 65
Formatting The Text 66
Other Formatting Rules 66
Adding A Spinner 67
Connecting Data To Other Fields On The Client-Side 68

Adding a DecimalTextBox 69

Properties for the DecimalTextBox 71
Getting And Setting The Value Properties 72
Editing Properties 74
Formatting Properties 76
Spinners Properties 77

CURRENCYTEXTBOX CONTROL 79

Features 80

Using the CurrencyTextBox 81
Getting and Setting the Value of the TextBox 82
Data Entry Validation 82
Formatting The Text 83
Other Formatting Rules 83
Adding A Spinner 84
Connecting Data To Other Fields On The Client-Side 85

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Adding a CurrencyTextBox **86**

Properties for the CurrencyTextBox **88**
Getting And Setting The Value Properties 89
Editing Properties 91
Formatting Properties 93
Spinner Properties 94

PERCENTTEXTBOX CONTROL **96**

Features **97**

Using the PercentTextBox **98**
Getting and Setting the Value of the TextBox 99
Data Entry Validation 99
Formatting The Text 100
Other Formatting Rules 100
Adding A Spinner 101
Connecting Data To Other Fields On The Client-Side 102

Adding a PercentTextBox **103**

Properties for the PercentTextBox **105**
Getting And Setting The Value Properties 106
Editing Properties 109
Formatting Properties 111
Spinner Properties 112

FILTEREDTEXTBOX CONTROL **114**

Features **115**

Using the FilteredTextBox **116**
Setting the Character Set 116

Adding a FilteredTextBox Control **117**

Properties of the FilteredTextBox **120**
Get and Set The Value Properties 120
Character Set Rules Properties 121

MULTISEGMENTDATAENTRY CONTROL **123**

Features **124**

Using the MultiSegmentDataEntry Control **125**
Defining Segments 126
Segment Validation Rules 126
Splitting and Joining Rules 126
Formatting The Segment 126
Getting and Setting the Value of the Control 127
Data Entry Validation 127
Validation on AutoPostBack 129
Interactive Hints 130
Setting Up Hints with PopupViews 130
Setting Up Hints in a Label or Panel 131

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Data Entry Rules 132
Changing the Appearance with Style Sheets 133

Adding a MultiSegmentDataEntry Control 134

Properties for the MultiSegmentDataEntry Control 137
Getting and Setting Values Properties 138
Segments Property 140
Behavior Properties 142
Tab Rules Properties 144
Appearance Properties 145
Hint Properties 147

Properties for the PeterBlum.DES.TextSegment Class 148
Get and Set Text Properties 148
Split and Join Text Properties 149
Data Entry and Validation Rule Properties 152
Appearance Properties 153
Behavior Properties 155
Hint Properties 156

Properties for the PeterBlum.DES.IntegerTextSegment Class 158
Get and Set Text Properties 158
Split and Join Text Properties 159
Data Entry and Validation Rules Properties 162
Appearance Properties 163
Behavior Properties 165
Spinners Properties 166
Hint Properties 167

Properties for the PeterBlum.DES.DropDownListSegment Class 169
Get and Set Text Properties 169
Split and Join Text Properties 170
DropDownList Items Properties 173
Data Entry and Validation Rules Properties 174
Appearance Properties 175
Behavior Properties 176
Hint Properties 177

Examples 179
U.S./Canada Phone Number 179
IP Address 180
Date 181

MultiSegmentDataEntryValidator Control 182
Using This Condition 182
Condition Properties for MultiSegmentDataEntryValidator 183

Subclassing MultiSegmentDataEntry 184

ADDITIONAL TOPICS FOR USING THESE CONTROLS 185

PAGE LEVEL PROPERTIES 186

Properties on PeterBlum.DES.Globals.Page 186
SpinnerManager Property 189

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

VALIDATION WITH THE NATIVE VALIDATION FRAMEWORK 191

Setting Up DES with the Native Validation Framework 192

Check the DataType with the CompareValidator Control 193
Using this Validator 193
Properties 193

Compare To Value with the CompareValidator Control 194
Using this Validator 194
Properties 194

Compare Two Fields with the CompareValidator Control 195
Using this Validator 195
Properties 195

RangeValidator Control 196
Using this Validator 196
Properties 196

DifferenceValidator Control 197
Using this Validator 197
Properties 197

MultiSegmentDataEntryValidator Control 199
Using This Validator 199
Properties for MultiSegmentDataEntryValidator 199

JAVASCRIPT SUPPORT FUNCTIONS 200

General Utilities 200

Getting and Setting the Value of Numeric TextBoxes 203

Validation Functions 205

MultiSegmentDataEntry Functions 206

ADDING YOUR JAVASCRIPT TO THE PAGE 211

Embedding the ClientID into your Script 211

Debugging Your JavaScript 212

TROUBLESHOOTING 213

Runtime Problems 213

Design Mode Problems 213

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

http://www.PeterBlum.com
For technical support and other assistance, see page 7

License Information

This document includes information for the Peter's TextBoxes module in Peter's Data Entry Suite. If you licensed the complete Suite or the Peter's TextBoxes module, you have all features found in this User's Guide, unless otherwise noted.

Platform Support

This product was written for Microsoft ASP.NET. It supports all versions from 1.0 up. It includes assemblies specific to ASP.NET 1.x and ASP.NET 2. It is compatible with all browsers, scaling down automatically when the browser has a limitation. In some cases, that means the control turns off its client-side functionality or turns itself off entirely.

This product is designed to scale properly even when the Page's **ClientTarget** property causes the HttpBrowserCapabilities (Request.Browser) to falsely state the browser. In other words, you can't fool these controls with an upLevel clientTarget. This is absolutely necessary because feeding the wrong browser will generate incorrect client side scripts giving the user's scripting errors. It was also considered a requirement to hide features that didn't work on the browser to give the user the best interface. For more, see "Browser Support" in the **General Features Guide**.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Technical Support and Other Assistance

PeterBlum.com offers free technical support. This is just one of the ways to solve problems. This section provides all of your options and explains how technical support is set up.

Troubleshooting Section of this Guide

This guide includes an extensive set of problems and their solutions. See "Troubleshooting". This information will often save you time.

Developer's Kit

The Developer's Kit is a free download that provides documentation and sample code for building your own classes with this framework. It includes:

- Developer's Guide - Overviews of each class with examples, step-by-step guides, and other tools to develop new classes.
- MSDN-style help file - Browse through this help file to learn about all classes and their members.
- Sample code in C# and VB.

You can download it from http://www.peterblum.com/DES/DevelopersKit.aspx.

PeterBlum.Com MessageBoard

Use the message board at http://groups.yahoo.com/groups/peterblum to discuss issues and ideas with other users.

Getting Product Updates

As minor versions are released (4.0.1 to 4.0.2 is a minor version release), you can get them for free. Go to http://www.PeterBlum.com/DES/Home.aspx. It will identify the current version at the top of the page. You can read about all changes in the release by clicking "Release History". Click "Get This Update" to get the update. You will need the serial number and email address used to register for the license.

As upgrades are offered (v4.0 to v4.1), PeterBlum.com will determine if there is an upgrade fee at the time. You will be notified of upgrades and how to retrieve them through email.

PeterBlum.com often adds new functionality into minor version releases.

Technical Support

You can contact Technical Support at this email address: Support@PeterBlum.com. I (Peter Blum) make every effort to respond quickly with useful information and in a pleasant manner. As the only person at PeterBlum.com, it is easy to imagine that customer support questions will take up all of my time and prevent me from delivering to you updates and cool new features. As a result, I request the following of you:

- Please review the Troubleshooting section first. See "Troubleshooting".
- Please try to include as much information about your web form or the problem as possible. I need to fully understand what you are seeing and how you have set things up.
- If you have written code that interacts with my controls or classes, please be sure you have run it through a debugger to determine that it is working in your code or the exact point of failure and error it reports.
- If you are subclassing from my controls, I provide the DES Developer's Kit that includes the Developers Guide.pdf, Classes And Types help file, and sample files. *I can only offer limited assistance as you subclass because this kind of support can be very time consuming*. I am interested in any feedback about my documentation's shortcomings so I can continue to improve it.
- I cannot offer general ASP.NET, HTML, style sheet, JavaScript, DHTML, DOM, or Regular Expression mentoring. If your problem is due to your lack of knowledge in any of these technologies, I will give you some initial help and then ask you to find assistance from the many tools available to the .Net community. They include:

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- Books
- www.asp.net forums and tutorials
- Microsoft's usenet newsgroups such as microsoft.public.dotnet.framework.aspnet. See http://groups.google.com/groups?hl=en&lr=&ie=UTF-8&group=microsoft.public.dotnet
- Google searches. (I virtually live in Google as I try to figure things out with ASP.NET.) http://www.Google.com. Don't forget to search the "Groups" section of Google!
- http://aspnet.4guysfromrolla.com/, http://www.dotnetjunkies.com, http://www.aspalliance.com/
- For DHTML, Microsoft provides an excellent guide at http://msdn2.microsoft.com/en-us/library/ms533050.aspx.
- For DOM, start with the DHTML guide. Topics that are also in DOM are noted under the heading "Standards Information"
- For JavaScript, I recommend http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference.

As customers identify issues and shortcomings with the software and its documentation, I will consider updating these areas.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

What does Peter's Data Entry Suite Do?

Peter's Data Entry Suite ("**DES**") is a suite of ASP.NET controls designed around the concepts of data entry: validation, entry fields, and interactive behaviors in response to a user's action. The suite includes several modules that can be purchased stand-alone. This User's Guide covers the data entry controls.

Peter's Data Entry Suite was designed to improve upon the form validation concept build into the .Net framework. It completely replaces Microsoft's original Validator controls, as they imposed serious limitations on how a Validator can look and act. As part of the work, extensive client-side JavaScript code was written. This JavaScript code lends itself well to other client-side tasks, such as formatting the entry of a CurrencyTextBox and setting focus to a field. Since validation is always part of data entry, **Peter's Data Entry Suite** addresses numerous other requirements of a good user interface for data entry.

Here are the major aspects of **Peter's Data Entry Suite**:

- Validation – The same idea as the concept introduced by Microsoft, with Validator controls to detect and report errors on the page. DES provides extensive enhancements over Microsoft's controls that allow your sites to have a more professional appearance with Validators and make it much easier to evaluate the data in your web form. Its rich feature set lets you set a few properties instead of writing custom code and hacks to work around the limitations of Microsoft's Validators. It includes 28 Validators and several other controls.
    - There are 11 Validators in **Peter's Professional Validation**, designed to handle common cases, like required fields, comparisons and textlength limits.
    - There are 14 Validators in **Peter's More Validators**, designed to handle more specialized situations like credit card numbers and duplicate entries amongst several fields.
    - There are two Validators designed for you to plug in your own validation logic, the CustomValidator and IgnoreConditionValidator
    - The MultiConditionValidator lets you combine the validation logic of other Validators into one boolean expression, often avoiding the use of writing code in CustomValidators.
    - The ValidationSummary control displays a consolidated list of all errors reported as the page is submitted.
    - The RequiredFieldMarker and RequiredFieldDescription controls standardize the user interface for indicating a field is required.
    - The CombinedErrorMessages control merges the error messages from several Validators to save screen real estate.
    - The LocalizableLabel control is an enhanced Label that supports localization. Localization is an important aspect to DES. Labels are optionally shown in the error messages.
    - A variety of button controls and ways to submit the page so that validation is run automatically.

  See the **Validation User's Guide**.

- TextBoxes – DES provides a number of controls as replacements to Microsoft's TextBox control.
    - Its own enhanced TextBox control, the basis for all other textboxes, introduces numerous common client-side tricks for TextBoxes.
    - The IntegerTextBox, DecimalTextBox, CurrencyTextBox, and PercentTextBox handle data entry of these numeric formats.
    - The FilteredTextBox limits entry to the character set of your choice.
    - The MultiSegmentDataEntry control combines multiple textboxes and dropdownlists together to handle a single field that has a distinctive pattern, such as phone numbers and credit card numbers. It is a great substitute for a masked textbox with a more powerful user interface.

  This **User's Guide** addresses these features.

- Date And Time – Controls for entry of date and time information.
    - DateTextBox – Date entry with a popup calendar
    - AnniversaryTextBox – Date entry without the year.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- MonthYearTextBox – Month and year entry
- TimeOfDayTextBox – Time of day entry
- DurationTextBox – Time duration entry
- Calendar – A very powerful replacement to the native ASP.NET calendar
- MonthYearPicker – Month and year entry using the mouse
- TimePicker – Time entry using the mouse
- Popup versions of Calendar, MonthYearPicker, and TimePicker
- Validators for these controls. There are versions for the DES Validation Framework and Native ASP.NET Validation Framework if you choose not to use the DES Validation Framework.

See the **Date And Time User's Guide**.

- Interactive Pages – There are numerous ways to make your web forms more interactive and user friendly through JavaScript. These techniques will make your web forms feel more like a window in a Windows application.
  - The FieldStateController monitors clicks and changes on a field and modifies other controls. It can modify almost any attribute of a field: visibility, enabled, style sheet class, value and more. For example, use it when you want a textbox to be disabled until the user marks a checkbox. There are four versions of the FieldStateController: FieldStateController, MultiFieldStateController, FSCOnCommand, and MultiFSCOnCommand.
  - The CalculationController lets you describe a calculation that uses the textboxes on the page. It can display the result of the calculation in a label or another textbox. Validators can validate the result of the calculation. For example, a RangeValidator can make sure the total of 3 textboxes is within 0 to 100.
  - Add a ContextMenu to your web forms, supplying javascript commands to regions and controls within the page.
  - The TextCounter monitors the number of keystrokes in a textbox, displaying the count and warning the user as they near and reach the limits.
  - The Interactive Hints system shows a hint as the user moves into a field. The hint can appear in a popup (floating element) or on the page in a Label or Panel. It can replace standard ToolTips and can appear in the browser's status bar too.
  - The ChangeMonitor watches for edits in the page. Until a change is made, selected buttons are disabled. After, they are enabled.
  - Enhanced Buttons with several javascript tricks including showing a confirmation message, disabling until data is changed (see the ChangeMonitor), and disabling on submit.
  - Direct the Enter key to click a specific button. Enhances DES's TextBoxes with the **EnterSubmitsControlID** property.

  See the **Interactive Pages User's Guide**.

- Input Security – Hackers attack your web site through its inputs – data entry controls, query strings, hidden fields, and cookies – to access your database (called SQL Injection) and modify your pages with scripts (called Script Injection or Cross-site scripting). These attacks can be very damaging to a business, destroying data, exposing private customer information, or exposing customers to content that you would never want on your site. Every public web site should be designed with a defense system. With Peter's Input Security, you have that defense system.

  Validators play an important role in blocking these attacks. However, they have their limitations. Peter's Input Security introduces specialized Validators to detect and block attacks. It also provides a "best practices" framework for protecting your site against attacks.

  See the **Input Security User's Guide**.

- General Features – The features throughout this product are supported by these controls and tools:
  - PageManager control – Each page has numerous settings. Use this control to apply those settings in without programming.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- Global Settings Editor – A stand-alone Windows application to customize numerous global settings used by DES.
- NativeControlExtender control – Extends various native controls to support DES Validation, Interactive Hints, the ChangeMonitor, and more.
- String Lookup System – A mechanism to set most string-type properties from data stored in resources or a database.
- LocalizableLabel control – The Label control enhanced to support the String Lookup System.

Throughout this product, one of the most important design concepts is to allow expansion through the object-oriented concepts of subclassing and delegation. This allows you to build your web site the way you want it. The controls are built upon several discrete object classes. Even the client-side JavaScript is designed for expansion. See the Developer's Kit to learn how to program with **Peter's Data Entry Suite**.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Peter's TextBoxes Overview

Peter's Data Entry Suite ("DES") focuses on enhancing data entry on your ASP.NET web forms. The TextBox control is one of the most dynamic and frequently used data entry controls. The textual data it gathers represents a wide variety of data types: free-form text, heavily patterned text (like a phone number), numbers, dates and more.

The TextBox control supplied with ASP.NET does little to handle unique data types. It doesn't filter out illegal characters. It requires you to write code to convert between the data type and its string representation. There are also numerous client-side techniques web page developers use, none of which are addressed with the ASP.NET TextBox.

DES has built enhanced TextBox controls and its MultiSegmentDataEntry control to help with heavily patterned text. Here are the features of these controls.

Enhanced TextBox Overview

The Enhanced TextBox is a descendant of the ASP.NET TextBox, and is intended to be used in all places where you now use the ASP.NET TextBox. Consider converting your existing forms to it or one of the other DES TextBoxes, as they all subclass from `PeterBlum.DES.TextBox`.

The Enhanced TextBox has several useful client-side extensions:

- The "Value When Blank" system lets you customize how blank textboxes are displayed. You can provide text like "Please fill this in" with the **ValueWhenBlank** property and a style sheet class to make blank fields stand out with the **ValueWhenBlankCssClass** property.
- The **DisableAutoComplete** property lets you omit the Autocomplete list from the textbox. There are cases where Autocomplete does not belong. Supported by several browsers.
- The **DisablePaste** property turns off the ability to paste on Internet Explorer and any other browser that supports the client-side onpaste event.
- The AutoPostBack feature now supports client-side validation of the textbox before submitting the page when using the **AutoPostBackValidates** property. This way, it only submits if the changed textbox is also valid.
- Auto tabbing to another field in three ways:
  1. Use the **TabAtMaxLength** property to tab when the text fills to the **MaxLength**.
  2. Use the **TabOnEnterKey** property to tab when the user types ENTER.
  3. Use the **TabOnTheseChars** to define a list of characters that tab when any are typed.
- Use the **EnterSubmitsControlID** property to specify a button that will be clicked when ENTER is typed. *This feature requires a license for the Peter's Interactive Pages.*
- Use the **Hint** property to display a hint as the user tabs into the field. The hint is removed as tab departs the field. *This feature requires a license for the Peter's Interactive Pages.*
- Smarter detection of "onchange" events. The client-side onchange event is a signal that the textbox has changed after focus leaves the field. Validator controls and the TextBoxes that autoreformat rely on the onchange event. However, browsers have two limitations that prevent them from firing the onchange event, even after a change is made:
  1. The Autocomplete feature on Internet Explorer does not cause it to fire. So users who pick from the Autocomplete list do not get the immediate feedback of validation.
  2. When JavaScript assigns a new value to the textbox, it does not cause it to fire.

  Enhanced TextBox installs code that always fires the onchange event when focus is lost and a change occurred.

In addition, it has these enhancements:

- The **TextAlign** property lets you choose between left, center, and right alignment.
- The **ToolTip** supports the String Lookup System through its **ToolTipLookupID** property. This provides localization on the ToolTip.

See "Enhanced TextBox Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

IntegerTextBox, DecimalTextBox, CurrencyTextBox, and PercentTextBox Overview

These four TextBoxes elegantly handle four numeric data types: Integers, Decimals, Currencies, and Percents. They all have these features:

- They subclass from `PeterBlum.DES.TextBox`, inheriting all of its qualities.
- They are designed around data entry for a particular data type.
- They are culture sensitive, respecting the CultureInfo object defined on the **PeterBlum.DES.Globals.Page.CultureInfo** property.
- On the client-side, they filter keystrokes so users can only enter characters suitable to the data type.
- They include server-side properties to get and set the text using the data type, avoiding you having to write conversion code. For example, an IntegerTextBox includes the property **IntegerValue** that supports integer values.
- On the client-side, they reformat if needed as the user exits the textbox. For example, if the user enters "43" into a CurrencyTextBox, it reformats to "43.00" (assuming the currency culture uses that format.)
- When you attach a DES-based Validator to them, the Validator automatically determines the correct data type.
- They offer a spinner control (up and down arrows to the right of the textbox) to increment the value.

Features unique to each TextBox are:

- IntegerTextBox – Accepts positive and negative integers. There is a property to only permit non-negative values.
- DecimalTextBox – Accepts positive and negative decimal values. There is a property to only permit non-negative values. It conforms to the culture of the page (which can be overridden.)
- CurrencyTextBox – Accepts positive and negative decimal values following a currency format. There is a property to only permit non-negative values. Another property determines if the currency symbol is accepted and shown. It conforms to the culture of the page (which can be overridden.)
- PercentTextBox – Handles integer and decimal percents. The percent symbol is optionally allowed.

See "IntegerTextBox Control", "DecimalTextBox Control", "CurrencyTextBox Control", and "PercentTextBox Control".

FilteredTextBox Overview

The `PeterBlum.DES.FilteredTextBox` prevents the user from entering invalid characters into a textbox. You specify the set of characters that are valid or invalid.

It does not apply a pattern to the characters entered. Users can enter any character from the set of characters at any location and for as many times as desired. Typically you will use a RegexValidator or the MultiSegmentDataEntry control to verify that the text matches a pattern.

The CharacterValidator intelligently configures itself to the settings you assign to the FilteredTextBox so that you don't have to do it twice.

Some common uses of this control are:

- Password definition
- Person's name (usually doesn't have punctuation or numbers)
- Phone number
- Credit card number

The `PeterBlum.DES.FilteredTextBox` inherits all of the features of the `PeterBlum.DES.TextBox`.

See "FilteredTextBox Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

MultiSegmentDataEntry Control Overview

Use the `PeterBlum.DES.MultiSegmentDataEntry` control as a substitute for a TextBox when you have a strongly patterned data type. It is a similar idea to a masked textbox, where each character position requires a specific character. For example, this control and masked textboxes are used to enter phone numbers, IP addresses, and dates (although the **Peter's Date And Time** module provides much better date entry with its own DateTextBox.)

While the masked textbox is one TextBox control with precise keyboard filtering, the MultiSegmentDataEntry control defines multiple TextBoxes or DropDownLists, one for each "segment" of the data where the user types. Any static text, like the period found between each segment of an IP address, is displayed between the segments and the user doesn't have to type it. This design has several advantages over the masked textbox:

- Browsers have a mixture of capabilities when it comes to handling typing at a particular position. To do it right, you need to know the start and end index of the insertion point (when they are different, they user has selected some text). Internet Explorer does not make this information available, although the Mozilla browsers do. Usually masked textboxes for Internet Explorer can allow illegal cases as the user moves the insertion point within the existing text. The MultiSegmentDataEntry control never has this problem. It does not need to know about the insertion point.
- Each segment's textbox can have its own character set. For example, one can allow letters while another allows digits.
- Segments can offer a DropDownList, which is a very good user interface for having a limited set of choices, like the months of the year.
- Individual segments know when you type a character that separates two segments, like the period between IP address segments. They autotab to the next segment so the user can enter the text naturally, without worrying about the tab key.
- TextBoxes can have a maximum length that provides additional guidance to the user. Plus they can autotab when the limit is hit.
- When working with integers, your textbox can offer spinners (up/down arrows) to change the value.
- Each segment can have its own Validator in addition to a master Validator for the entire text. For example, an IP address needs a RangeValidator for values from 0 to 255 on each segment.
- Hints can be shown on the page as focus moves into a segment. So on-screen documentation is available.

All of these features help greatly improve the user's experience so the user knows what to do and understands how to enter patterned data without knowing the pattern in advance.

To make it work, the MultiSegmentDataEntry control has the ability to get and set single patterned string, splitting or joining it according to rules that you specify. For example, on a phone number, the dash character is just formatting and the digits found before and after a dash appear in different segments.

The MultiSegmentDataEntry control supports most of the existing Validators that evaluate textboxes. For example, if you set it up for credit card numbers, you can use the CreditCardNumberValidator on it. Additionally, since each segment has rules like text length, valid characters, and "requires text", the MultiSegmentDataEntryValidator validates any pattern.

See "MultiSegmentDataEntry Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Date and Time Entry TextBoxes

The **Peter's Date And Time** module also provides some textboxes within the **Peter's Data Entry Suite**. They are built around date and time entry: DateTextBox, TimeOfDayTextBox, DurationTextBox, AnniversaryTextBox, and MonthYearTextBox.

See **Peter's Date And Time User's Guide**.

Other Data Entry Controls

While HTML only defines a small group of data entry controls (inputs, textarea, select), developers have created a rich collection of data entry tools, such as the ASP.NET calendar and RichTextBoxes. Peter's Data Entry Suite only includes the controls described above. However, it works well with most other controls because ASP.NET's webcontrol concept allows mixing controls from various sources.

Some notable data entry controls:

- Rich text boxes, which are textboxes that allow WYSIWIG entry, are available from several third parties. See http://www.411asp.net/home/assembly/contentm.
- ComboBoxes, which are more powerful DropDownLists, are available from several third parties. The DES Validation Framework includes support for EasyListBox and RadComboBox. See http://www.411asp.net/home/assembly/datalist

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Enhanced TextBox Control

The Enhanced TextBox control (`PeterBlum.DES.TextBox`) is a substitute for the TextBox control included with ASP.NET. It is a subclass of `System.Web.UI.WebControls.TextBox`. So it is fully compatible with all code that already uses the ASP.NET TextBox. Due to its improved featureset, you are encouraged to use it wherever you use the ASP.NET textbox. However, this is not a requirement. The rest of DES can use either the ASP.NET TextBox or `PeterBlum.DES.TextBox`.

All other DES TextBoxes subclass from the Enhanced TextBox.

Click on any of these topics to jump to them:

- Features
- Using the Enhanced TextBox Control
  - Getting and Setting the Value
  - When the TextBox is Empty
  - Validation on AutoPostBack
  - Interactive Hints
  - AutoComplete and "Smart Change System"
  - Other Behaviors
- Adding an Enhanced TextBox
- Converting the ASP.NET TextBox to the Enhanced TextBox
- Properties for the Enhanced TextBox

If your data is one of these types, consider these alternatives to the TextBox:

Integer	IntegerTextBox Control
Double or Decimal	DecimalTextBox Control
Currency (using Double or Decimal type)	CurrencyTextBox Control
Percentage (using Integer, Double, or Decimal)	PercentTextBox Control
Textual with a specific character set	FilteredTextBox Control
Textual with a strong pattern (where a "mask" would apply)	MultiSegmentDataEntry Control
Date	DateTextBox, AnniversaryTextBox, or MonthYearTextBox. See the **Date and Time User's Guide**.
Time of day	TimeOfDayTextBox. See the **Date and Time User's Guide**.
Duration	DurationTextBox. See the **Date and Time User's Guide**.
Fixed list of strings	DropDownList or ListBox (from ASP.NET controls) or combobox from a third party such as EasyListBox and RadControls.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Features

Enhanced TextBox (`PeterBlum.DES.TextBox`) is a descendant of the ASP.NET TextBox, and is intended to be used in all places where you now use the ASP.NET TextBox. Consider converting your existing forms to it or one of the other DES TextBoxes, as they all subclass from `PeterBlum.DES.TextBox`.

The Enhanced TextBox has several useful client-side extensions:

- The "Value When Blank" system lets you customize how blank textboxes are displayed. You can provide text like "Please fill this in".
- Supports the Interactive Hints feature from **Peter's Interactive Pages**. Use the **Hint** property to display a hint as the user tabs into the field. The hint is removed as tab departs the field. *This feature requires a license for the Peter's Interactive Pages.*
- Supports the Enhanced ToolTips feature from **Peter's Interactive Pages**. *This feature requires a license for the Peter's Interactive Pages.*
- The **AutoPostBack** property has been extended to support client-side validation when using the DES Validation Framework. It prevents a postback when there is an error.
- The **DisableAutoComplete** property lets you omit the Autocomplete list from the textbox. There are cases where Autocomplete does not belong. Supported by several browsers.
- The **DisablePaste** property turns off the ability to paste on Internet Explorer and any other browser that supports the client-side onpaste event.
- Auto tab to another control in several ways:
    - Use the **TabAtMaxLength** property to tab when the text fills to the **MaxLength**.
    - Use the **TabOnEnterKey** property to tab when the user types ENTER.
    - Use the **TabOnTheseChars** to define a list of characters that tab when any are typed.
    - Use the **TabByArrowKeys** to advance to the next or previous field when the left and right arrow keys hit the end of text.
    - Use the **TabOnBackspace** to tab to a previous field when the textbox is empty and the user types BACKSPACE.
- Use the **EnterSubmitsControlID** property to specify a button that will be clicked when ENTER is typed. *This feature requires a license for the Peter's Interactive Pages.*
- Smarter detection of client-side "onchange" events. The client-side onchange event is a signal that the textbox has changed after focus leaves the field. Validator controls and the TextBoxes that autoreformat rely on the onchange event. However, browsers have two limitations that prevent them from firing the onchange event, even after a change is made:
    - The browser's Autocomplete feature does not cause it to fire. So users who pick from the Autocomplete list do not get the immediate feedback of validation.
    - When JavaScript assigns a new value to the textbox, it does not cause it to fire.

  Enhanced TextBox installs code that always fires the onchange event when focus is lost and a change occurred.
- The **TextAlign** property lets you choose between left, center, and right alignment.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the Enhanced TextBox Control

As a direct subclass of `System.Web.UI.WebControls.TextBox`, you should use the documentation of that control for many of its features including setting styles, the **Text** property, the **AutoPostBack** property and the **TextChanged** event. See System.Web.UI.WebControls.TextBox Members for its complete list of properties, methods, and events.

Click on any of these topics to jump to them:

- Getting and Setting the Value
- When the TextBox is Empty
- Validation on AutoPostBack
- Interactive Hints
- AutoComplete and "Smart Change System"
- Other Behaviors

Getting and Setting the Value of the TextBox

When getting or setting the value, use the **Text** property.

When you need a server side event that is fired when the textbox's value has changed, use the **TextChanged** event.

Data Entry Validation

Consider validation a manditory part of data entry. It prevents illegal entries from getting into your database. Also protect yourself by setting up server side validation. Hackers often turn off javascript in their browser in hopes that your server side code doesn't protect against their illegal data.

See the **Validation User's Guide**.

DES Validation Framework Guidelines

Determine if your textbox's data needs some kind of validation:

- Are there any illegal characters? Use the CharacterValidator. (Also switch to the FilteredTextBox Control.)
- Is this data a candidate for a SQL Injection or Cross Site Scripting attack? Use the PageSecurityValidator or FieldSecurityValidator. See **Input Security User's Guide** for details.
- Is there a pattern to your data? Use the RegexValidator with an expression that confirms your pattern. Here is a great site for popular patterns: http://regexlib.com.
- Is there a fixed list of values? Use the CompareToStringsValidator.
- Is there a minimum or maximum size restriction? Use the TextLengthValidator or TextLengthSecurityValidator.

Always set up server side validation. Test **PeterBlum.DES.Globals.Page.IsValid** in your postback event handler methods. Only use the data if it is `true`.

Native Validation Framework Guidelines

Determine if your textbox's data needs some kind of validation:

- Are there any illegal characters? Use a regular expression like this within a RegularExpressionValidator.

  ```
  ^[legal characters here]*$
  ^[^illegal characters here]*$
  ```

  See http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference:Global_Objects:RegExp for regular expression assistance.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Examples:

Only letters	^[A-Za-z]*$
Only digits	^[0-9]*$ or ^[\d]*$
Letters, digits, underscore	^[A-Za-z\d_]*$ or ^[\w]*$
Letters, digits, underscore, space, period	^[\w .]*$ (there is a space character between \w and period.)
All except letters	^[^A-Za-z]*$
All except space and period	^[^ .]*$ (there is a space character between ^ and period.)

- Is this data a candidate for a SQL Injection or Cross Site Scripting attack? Build server side defenses to neutralize the attack. See **Input Security User's Guide** for details.
- Is there a pattern to your data? Use the RegexValidator with an expression that confirms your pattern. Here is a great site for popular patterns: http://regexlib.com.
- Is there a fixed list of values? Use the RegexValidator with a pattern like this:

  ```
  ^((value1)|(value2)|(value3))$
  ```

  A common error is to include characters that are special symbols in these values without demarking them with a lead \ character. These characters should have a lead slash: period (.), comma (,), ?, !, \, ^, $, *, -, +, left and right parenthesis, brackets, curly braces,
- Is there a minimum or maximum size restriction? Use the RegularExpressionValidator with this expression:

  ```
  ^[\w\W]{minimum,maximum}$
  ^[\w\W]{maximum}$
  ^[\w\W]{minimum,}$
  ```

  where minimum is an integer for the minimum value and maximum is an integer for the maximum value.

  For example, a maximum of 40: ^[\w\W]{40}$

Always set up server side validation. Test **Page.IsValid** in your postback event handler methods. Only use the data if it is `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

When the TextBox is Empty

Blank textboxes can show special text and/or a style sheet. Use the **ValueWhenBlank** property to provide text like "Please fill this in". Use the **ValueWhenBlankCssClass** property to change the a style sheet class. As the textbox gets focus, these values can be removed based on the **PeterBlum.DES.Globals.Page.ValueWhenBlankMode** property. When focus leaves, if the field is still blank, the text and style sheet class are restored.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Validation on AutoPostBack

If you use **AutoPostBack**, it now automatically validates before posting back except when **AutoPostBackValidates** is `false`. This avoids posting back when there is a validation error. It can either evaluate the validators assigned to the textbox are those in a specific group. When **CausesValidation** is `false` (requires ASP.NET 2 or higher), it evaluates the validators assigned to the textbox. When **CausesValidation** is `true`, it evaluates validators identified by the **ValidationGroup** property.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Interactive Hints

Note: Requires a license for the Peter's Interactive Pages module.

You can show a hint on the page as the user tabs into the textbox. A hint is similar to a tooltip. However, a tooltip only appears if the mouse is over the control. That is not the best way to communicate to the user as they are working in a textbox.

See the "Interactive Hints" section of the **Interactive Pages User's Guide** for details.

Assign your hint text to the **Hint** property. If you are using a PopupView, it optionally offers a Help button which can show additional text. That additional text is assigned to the **HintHelp** property.

The format of hints is determined by a `PeterBlum.DES.HintFormatter` object. You can either define one specific to this control in the **LocalHintFormatter** property or specify the name of one shared by other controls in the **SharedHintFormatterName** property. The HintFormatter determines where the hint is shown:

- PopupView or Label control. A PopupView is similar to a ToolTip, created with HTML and Javascript to float near the control. It can be dragged and closed. It can be customized with style sheets, images, and settings using the **Global Settings Editor**.
- In a tooltip
- In the browser's status bar

Most of the work is done by creating a HintFormatter object. The `PeterBlum.DES.HintFormatter` class describes how the hint text will be displayed. It provides its name, display mode - on the page or in a PopupView, if it's also in the tooltip and/or status bar, and more.

The HintManager object provides many page level properties that support this feature. (**HintManager** is a property of **PeterBlum.DES.Globals.Page** and the PageManager control.) See the "Interactive Hints" section of the **Interactive Pages User's Guide** for details.

Click on any of these topics to jump to them:

- Setting Up Hints with PopupViews
- Setting Up Hints in a Label or Panel

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Setting Up Hints with PopupViews

1. Set the text of the hint in the **Hint** property. It can contain HTML tags if desired. If you are using the same text in the **ToolTip** property, you do not need to assign anything to **Hint**. It uses the **ToolTip** property when **Hint** is "" unless you set the **HintManager.ToolTipsAsHints** property to `False`.
2. If you are using the **PopupView.HelpBehavior** property, set the **HintHelp** property to the appropriate text, whether it is a more detailed description, a title, a URL, or a script.
3. If you also want to show validation error messages (from the DES Validation Framework) in the PopupView, use the **HintManager.HintsShowErrors** property. See the "Interactive Hints" section of the **Interactive Pages User's Guide** for details.
4. Review the available PopupView definitions. PopupView definitions are created and edited within the "PopupView definitions for Hints" section of the **Global Settings Editor**. Each has a name, style sheets, images, width, and other behaviors.
5. Pick a PopupView definition. Usually the width differs depending on the size of the Hint text.
6. Assign the PopupView using one of these three approaches:
   - If you don't need the hint shown in the browser's status bar, just set the **SharedHintFormatterName** to the name of the Popup View. *DES automatically creates a HintFormatter for you with HintFormatter.DisplayMode and HintFormatter.PopupViewName correctly set.*
   - You will need to use a HintFormatter object. If you can use the same PopupView definition name and HintFormatter properties for several controls on this page, add a HintFormatter object to the **HintManager.SharedHintFormatters** property. This can be done in the PageManager control or programmatically.
     - Set the **HintFormatter.PopupViewName** to the name of the PopupView.
     - Set the **HintFormatter.DisplayMode** to `Popup`.
     - Consider if these properties apply: **InToolTip**, **InStatus**, and **TextFunctionName**. (All others are used when **Display** mode is not set to `Popup`.)

     See the "Interactive Hints" section of the **Interactive Pages User's Guide** for details.
   - Otherwise, use the LocalHintFormatter property on the control:
     - Set the **HintFormatter.PopupViewName** to the name of the PopupView.
     - Set the **HintFormatter.DisplayMode** to `Popup`.
     - Consider if these properties apply: **InToolTip**, **InStatus**, and **TextFunctionName**. (All others are used when **Display** mode is not set to `Popup`.)

     See the "Interactive Hints" section of the **Interactive Pages User's Guide** for details.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Setting Up Hints in a Label or Panel

1. Determine what kind of appearance that you want for your hint. It can be simply a Label or a Panel whose formatting encloses a Label and is fully hidden when there is no hint text to show. See the previous topic.
2. Determine the locations for hints. You can have one on the page, one for each group of controls, or even one for each control. When you put one next to a control, it can be located where Validators appear as there is a feature to prevent the hint from showing when a Validator is shown.
3. Add the controls for hints to the page. Remember that they will be hidden until focus is set to them.
4. If you are using a Panel that contains a Label, make sure the Label's ID is Panel.ID + "_Text".
5. Determine whether you need a HintFormatter object for this control or one can be shared amongst several controls. When using one specific to this control set up the HintFormatter object using the **LocalHintFormatter** property.
6. Otherwise, create the HintFormatter object in the **HintManager.SharedHintFormatters** property.
7. Set the **SharedHintFormatterName** property. When using **HintManager.SharedHintFormatters**, it should be the name of the HintFormatter object or "{DEFAULT}" to use your global default. Otherwise it should be "".
8. Assign the Panel or Label control to the property **HintFormatter.HintControlID**.
9. Set the **HintFormatter.DisplayMode** to `Static` or `Dynamic`. `Static` will preserve the space of the Panel or Label when it is hidden. `Dynamic` will not use any space when hidden.
10. If the Panel or Label appears at the same location as some Validators for this textbox, set **HintFormatter.HiddenOnError** to `true`.
11. If you also want the hint text to appear in the status bar, set **HintFormatter.InStatusBar** to `true`.
12. Set the text of the hint in the **Hint** property. It can contain HTML tags if desired. If you are using the same text in the **ToolTip** property, you do not need to assign anything to **Hint**. It uses the **ToolTip** property when **Hint** is "" unless you set the **HintManager.ToolTipsAsHints** property to `False`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

AutoComplete and "Smart Change System"

Without changing any properties, this textbox will immediately introduce the "smart change system". This technology fires the onchange event at times when the ASP.NET textbox won't: after Internet Explorer's AutoComplete feature is used and after changing the value. Since the onchange event is used with client-side validation, it will immediately make DES's Validators fire after AutoComplete. You can also turn off the AutoComplete feature by setting the **DisableAutoComplete** property to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Other Behaviors

Sometimes you do not want a field to allow pasting. For example, if the user is supposed to enter an email address twice to confirm it is correct. The second email address textbox should not permit pasting an email address copied from the first field. Set **DisablePaste** to `true`.

There are times when you want certain actions to tab to another field. DES provides several options for auto-tabbing. First set the control to receive focus with the **NextControlID** property. Then use any or all of these properties to establish the desired behavior: **TabAtMaxLength**, **TabOnEnterKey**, and **TabOnTheseKeys**. You can also auto-tab backward with the **TabOnBackspace** property. Set the control to receive focus in the **PreviousControlID** property.

You can make the ENTER key click a button by setting the button's control ID in the **EnterSubmitsControlID** property.

Note: EnterSubmitsControlID requires a license for the Peter's Interactive Pages.

If you want to add your own JavaScript code to either of the onkeypress or onkeydown events, use the **CustomKeyPressFunctionName** or **CustomKeyDownFunctionName** properties because the TextBox already uses those events.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Adding an Enhanced TextBox

These steps ask you to jump around the document using clicks on links. Adobe Reader offers a **Previous View** command to return to the link. Look for this in the Adobe Reader (shown v6.0)

1. Prepare the page for DES controls. See "Preparing a page for DES controls" in the **General Features Guide**. It covers issues like style sheets, AJAX, and localization.
2. Add a TextBox control to the page.

Visual Studio and Visual Web Developer Design Mode Users

Drag the TextBox control from the Toolbox onto your web form. *Be sure to select DES's TextBox, not System.Web.UI.WebControls.TextBox.*

Text Entry Users

Add the control (inside the <form> area):

```
<des:TextBox id="[YourControlID]" runat="server" />
```

Programmatically creating the TextBox control

- Identify the control which you will add the TextBox control to its **Controls** collection. Like all ASP.NET controls, the TextBox can be added to any control that supports child controls, like Panel, User Control, or TableCell. If you want to add it directly to the Page, first add a PlaceHolder at the desired location and use the PlaceHolder.
- Create an instance of the TextBox control class. The constructor takes no parameters.
- Assign the **ID** property.
- Add the TextBox control to the **Controls** collection.

In this example, the TextBox is created with an **ID** of "TextBox1". It is added to PlaceHolder1.

[C#]

```
PeterBlum.DES.TextBox vTextBox = new PeterBlum.DES.TextBox();
vTextBox.ID = "TextBox1";
PlaceHolder1.Controls.Add(vTextBox);
```

*Note: The namespace for these controls is PeterBlum.DES. If you prefer, add a **using** clause to that namespace on your form.*

[VB]

```
Dim vTextBox As PeterBlum.DES.TextBox = New PeterBlum.DES.TextBox()
vTextBox.ID = "TextBox1"
PlaceHolder1.Controls.Add(vTextBox)
```

*Note: The namespace for these controls is PeterBlum.DES. If you prefer, add an **Imports** clause to that namespace on your form.*

Guidelines for setting properties

- Design mode users can use the Properties Editor or the Expanded Properties Editor. (See "Expanded Properties Editor" in the **General Features Guide**.) The SmartTag ▶ also offers some of the most important properties.
- Text entry users should add the properties into the `<des:ControlClass>` tag in this format: `propertyname="value"`
- When setting a property programmatically, have a reference to the control's object and set the property according to your language's rules.

3. Set the properties associated with the TextBox. See "Properties for the Enhanced TextBox".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

4. Assign Validators to the TextBox. Validators are an important part of data entry, guiding the user and protecting you against hackers. See "Data Entry Validation".

5. Get and set the value using the **Text** property.

   Usually you will set the initial value in the `Page_Load()` method when **Page.IsPostBack** is `false`. You will get the submitted value in your post back event method, after checking that **PeterBlum.DES.Globals.Page.IsValid** (DES validation framework) or **Page.IsValid** (native validation framework) is `true`.

6. Here are some other considerations:

   - If you are using an AJAX system to update this control, set the **InAJAXUpdate** property to `true`. Also make sure the PageManager control or AJAXManager object has been setup for AJAX. See "Using these Controls With AJAX" in the **General Features Guide**. *Failure to follow these directions can result in incorrect behavior and javascript errors.*
   - This control does not preserve most of its properties in the ViewState, to limit its impact on the page. If you need to use the ViewState to retain the value of a property, see "The ViewState and Preserving Properties for PostBack" in the **General Features Guide**.
   - If you encounter errors, see the "Troubleshooting" section for extensive topics based on several years of tech support's experience with customers.
   - See also "Additional Topics for Using These Controls".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Converting the ASP.NET TextBox to the Enhanced TextBox

It takes very little to convert the `System.Web.UI.WebControls.TextBox` to `PeterBlum.DES.TextBox`. After all, DES's TextBox is a subclass of the other, so it is fully compatible. Your task is to change the namespace. You can either follow the steps below or run the **Web Application Updater** program with its option "Convert native controls".

1. In the ASP.NET Text, change `<asp:TextBox>` to `<des:TextBox>` and `</asp:TextBox>` to `</des:TextBox>`.
2. Make sure this line is at the top of the webform or user control:

   ```
   <%@ Register TagPrefix="des" Namespace="PeterBlum.DES"
   Assembly="PeterBlum.DES" %>
   ```

3. In any code that declares the TextBox, change the namespace: `System.Web.UI.WebControls.TextBox` to `PeterBlum.DES.TextBox`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties for the Enhanced TextBox

This control is subclassed from `System.Web.UI.Web.Controls.TextBox` and inherits all of the members in System.Web.UI.WebControls.TextBox Members. A few of the listed properties will be restated below.

Click on any of these topics to jump to them:

- Getting And Setting the Value Properties
- Editing Properties
- Behavior Properties
- Appearance Properties
- AutoPostBack Properties
- Value When Blank Properties
- ToolTip Properties
- Hint Properties
- Tab Rules Properties
- Client-Side Functions Properties

Getting And Setting the Value Properties

The Properties Editor shows these properties in the "Data" category.

- **Text** (string) – Gets and sets the value of this control. It is a string. See System.Web.UI.WebControls.TextBox.Text Property.

  Usually you will set the initial value in the `Page_Load()` method when **Page.IsPostBack** is `false`. You will get the submitted value in your post back event method, after checking that **PeterBlum.DES.Globals.Page.IsValid** (DES validation framework) or **Page.IsValid** (native validation framework) is `true`.

  If you need to convert between a string and another data type, consider these options:

  - Integer – Use the IntegerTextBox. Get and set your integer on its **IntegerValue** property.
  - Decimal – Use the DecimalTextBox. Get and set your integer on its **DoubleValue** property.
  - Currency – Use the CurrencyTextBox. Get and set your integer on its **DoubleValue** property.
  - Date – Use the DateTextBox in **Peter's Date and Time** module. Get and set the DateTime object on its **DateValue** property.
  - Time - Use the TimeOfDayTextBox in **Peter's Date and Time** module. Get and set the DateTime object on its **DateTimeValue** property.

- **TextChanged** (event) – This event is fired on post back when the TextBox value has changed. See System.Web.UI.WebControls.TextBox.TextChanged Event.

  Note: This event only works when the ViewState is enabled on the TextBox.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Editing Properties

The Properties Editor shows these properties in the "Editing" category.

- **TextMode** (enum System.Web.UI.WebControls.TextBoxMode) – Determines if the textbox is singleline, multiline, or a password. See "System.Web.UI.WebControls.TextBox.TextMode Property". It defaults to `SingleLine`.
- **MaxLength** (Integer) – Establishes a maximum number of characters that the user can type into the textbox. It is not supported when **TextMode** = `MultiLine` because the associated `<textarea>` tag does not offer any size constraint attributes. Use a TextLengthValidator on a Multiline TextBox.

  When 0, no limit is established. It defaults to 0.
- **ReadOnly** (Boolean) – When `true`, the control uses a ReadOnly state, where the user cannot change the text but can tab into the field. While **Enabled** = `false` never permits posting back data, **ReadOnly** does. So if you use client-side scripts to update the TextBox, choose this instead of **Enabled**. It defaults to `false`.
- **ConvertCase** (enum PeterBlum.DES.ConvertCase) – Converts the case of letters as they are typed on the client side. If client-side code is not running, the server side will still apply the correct case.

  The enumerated type `PeterBlum.DES.ConvertCase` has these values:

  - `No` - Do not change the text
  - `Upper` - Change letters to uppercase
  - `Lower` - Change letters to lowercase

  It defaults to `ConvertCase.No`.

  Browsers use this feature in different ways. Internet Explorer is the only browser that replaces the characters as you type. Other browsers update the textbox when focus is lost. Those without client-side scripting support will still apply the desired type on the server side so that the **Text** property accurately reflects this property's rule.
- **DisableAutoComplete** (Boolean) – Several browsers provide an "autocomplete" or "autofill" feature, where a list of previous entries appears as the user starts typing. This behavior often is inappropriate as the browser is guessing what the items are and its guess may be incorrect. For example, an integer textbox may still popup a list containing alphabetic entries. These browsers offer the ability to disable autocomplete on a field-by-field basis. Set this to `true` to disable it on this control. It defaults to `false`.
- **AutoCompleteType** (System.Web.UI.WebControls.AutoCompleteType) – See http://msdn2.microsoft.com/en-us/library/system.web.ui.webcontrols.textbox.autocompletetype(VS.80).aspx.
- **DisablePaste** (Boolean) – When you prefer that the user cannot paste anything into the textbox, set this to `true`. It is supported on Internet Explorer and any other browser that supports the 'onpaste' event. It defaults to `false`.
- **EnterSubmitsControlID** (string) – Use this when you want the ENTER key to click a specific button. The browser already has rules for clicking a button when you type ENTER. That button usually has a special border to identify it to the user. This property will override those rules. Here are cases where you will use **EnterSubmitsControlID**:

  - Suppose that you have two groups of fields, each with its own submit button. Each textbox should use this to point to its own submit button.
  - Internet Explorer for Windows has the following strange behavior: if you have only one data entry control, Internet Explorer submits the page without clicking the button first, causing it to skip any client-side validation.

  Assign the ID of the submit control. It must be assigned to a control in the same or a parent naming container. If the control is in another naming container, use **EnterSubmitsControl**.

  This feature fires the `click()` method on the client-side control. `click()` automatically runs the control's client-side `onclick` event. In the case of a submit control, it submits the page after firing client-side validation. There are a lot of controls that support `click()`, although they vary by browser. In addition to Buttons and ImageButtons, typical cases are hyperlinks, LinkButtons, checkboxes and radiobuttons. However, browsers don't all support the `click()` method on the same control. Here are the differences:

  - Internet Explorer and Opera 7 support it on hyperlinks (and LinkButton) while Mozilla and Safari do not.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- All support checkboxes and radiobuttons. However, Mozilla always removes the focus from the current field even if you don't set this feature up to move the focus (the focus is gone, not moved)
- All support Buttons the same way. This is the best choice for a control to click.

It defaults to "".

License Note: This property requires a license for the Peter's Interactive Pages.

- **EnterSubmitsControl** (System.Web.UI.Control) – This is an alternative to **EnterSubmitsControlID**. It has the same features as **EnablerSubmitsControlID**. It is assigned a reference to a control instead of an ID. As a result, it supports controls in any naming container. It must be assigned programmatically.

  When programmatically assigning properties to a TextBox control, if you have access to the submit control object, it is better to assign it here than assign its ID to the **EnterSubmitsControlID** property because DES operates faster using **EnterSubmitsControl**.

  License Note: This property requires a license for the Peter's Interactive Pages.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Behavior Properties

The Properties Editor shows these properties in the "Behavior" category.

- **ChangeMonitorGroups** (string) – When using the Change Monitor, the group names defined here are marked changed when this control is edited. See "Change Monitor" in the **Interactive Pages User's Guide**.

  The value of "" is a valid group name.

  For a list of group names, use the pipe character as a delimiter. For example: "GroupName1|GroupName2". If one of the groups has the name "", start this string with the pipe character: "|GroupName2".

  Use "*" to indicate all groups apply.

  It defaults to "".

- **InAJAXUpdate** (Boolean) – When using AJAX on this page, set this to `true` if the control is involved in an AJAX update. See "Using These Controls with AJAX" in the **General Features Guide**. It defaults to `false`.

- **Enabled** (Boolean) – When `false`, the control appears disabled and does not accept modifications. It defaults to `true`.

- **Visible** (Boolean) – When `false`, no HTML is output. This control is entirely unused. It defaults to `true`.

- **DataTypeCheckReportsRangeErrors** (Boolean) – *Only on numeric textboxes with MinValue and MaxValue properties.* When a numeric textbox has **MinValue** or **MaxValue** properties assigned, data entry can produce either a 'data type check' error or an 'out of range' error.

  Sometimes an out of range value can be considered a data type check error. For example, because a percentage is usually a positive number, you might set **MinValue** to 0. Reporting "-10" as an illegal value will make sense to the user without saying "The value -10 is out of range." More often, it is better to evaluate the range condition separately with a RangeValidator to let the user know explicitly by an error message that "The value is out of range."

  If you consider an out of range error to be just a data type check error and the DataTypeCheckValidator's error message is sufficient, set **DataTypeCheckReportsRangeErrors** to `true` and omit the RangeValidator. Then the DataTypeCheckValidator, DataTypeCheckCondition, and CompareValidator will all report an error when the value entered is a valid type but out of range.

  When **DataTypeCheckReportsRangeErrors** is `false`, use a RangeValidator to report an out of range error.

  It defaults to `false`.

- **ViewStateMgr** (PeterBlum.DES.ViewStateMgr) – Enhances the ViewState on this control to provide more optimal storage and other benefits. Normally, the properties of this control and its segments are not preserved in the ViewState. Just call `ViewStateMgr.TrackProperty("propertyname")` to record the property in the ViewState.

  For more details, see "The ViewState and Preserving Properties for PostBack" in the **General Features User's Guide**.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Appearance Properties

The Properties Editor shows these properties in the "Appearance" category.

- **BackColor**, **BorderColor**, **BorderStyle**, **BorderWidth**, **Columns**, **CssClass**, **Font**, **ForeColor**, **Height**, **Style**, **TabIndex**, and **Width** – These properties are described in System.Web.UI.WebControls.TextBox Members.

  Recommendation: Create a style sheet class and assign it to the **CssClass** property. Generally you only assign **Height** and **Width** on a case-by-case basis (although they can be put into style sheet classes).

- **TextAlign** (enum PeterBlum.DES.TextAlign) – By default, text is left justified in western cultures. Often users like to right justify numeric values in textboxes, especially when stacking them. This property offers justification. It adds the attribute `style='text-align:value;'` to the `<input type='text'>` tag.

  Some browsers do not support the text-align style and will ignore this property.

  The enumerated type `PeterBlum.DES.TextAlign` has these values:

  - `Default` – This is the default. When set to `Default`, no style=text-align attribute is written, allowing the style sheets of the page to manage it.
  - `Left`
  - `Center`
  - `Right`
  - `Justify`

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

AutoPostBack Properties

AutoPostBack allows an edit to immediately submit the page so that you can make changes to the page.

The Properties Editor shows these properties in the "AutoPostBack" category.

- **AutoPostBack** (Boolean) – When `true`, a change on the client-side will immediately submit the page. Usually you use this with the **TextChanged** event to update some part of the page and redraw it with that change. Use the **AutoPostBackValidates** or **CausesValidation** property to determine if it validates first. Use the **AutoPostBackTracksFocus** property to preserve the focus on the last control with focus. It defaults to `false`.

  Note: When using AJAX, the AJAX framework automatically makes a callback instead of a postback.

- **AutoPostBackValidates** (Boolean) – When `true` and **AutoPostBack** is `true`, before submitting, it validates the textbox. If there are any errors, it does not submit. This avoids post back when the field has errors. It defaults to `true`.

- **CausesValidation** (Boolean) – *ASP.NET 2.0 Only*. When `true` and **AutoPostBack** is `true`, before submitting, it evaluates all Validators in the validation group assigned to the **ValidationGroup** property. It is an alternative to **AutoPostBackValidates**, which only evaluates the Validators assigned to this textbox. When **CausesValidation** is `true`, **AutoPostBackValidates** is ignored. It defaults to `false`.

- **ValidationGroup** (string) – *ASP.NET 2.0 Only*. When **CausesValidation** and **AutoPostBack** are `true`, AutoPostBack first validates all Validators in the validation group defined by this property. It defaults to "" (which is a valid validation group name.)

- **AutoPostBackTracksFocus** (Boolean) – *ASP.NET 2.0 Only*. When **AutoPostBack** and **AutoPostBackTracksFocus** are both `true`, the page will attempt to preserve the focus on the last field with focus before the page was submitted. It defaults to `false`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Value When Blank Properties

These properties establish how the textbox looks when it is blank. As focus is set on the textbox, you can have the look change to its normal appearance using the **PeterBlum.DES.Globals.Page.ValueWhenBlankMode** property. As focus leaves the textbox, if it is still blank it will resume using these properties.

The Properties Editor shows these properties in the "Value When Blank" category.

- **ValueWhenBlank** (string) – When the TextBox is empty, it is reassigned to this value. For example, "Please fill in this field". On the client-side, as focus is set to this field, the value will be restored to empty if the **PeterBlum.DES.Globals.Page.ValueWhenBlankMode** property is `RemoveText` or `RemoveBoth`. As focus leaves this field, when the field value is "", **ValueWhenBlank** is reassigned.

  When this property is "", no changes are made to the value of the TextBox.

  It defaults to "".

  When the field is posted back, if the value posted back was **ValueWhenBlank**, the **Text** property will return "". So this value is hidden from you and is only used to communicate with the user. DES's Validators will also recognize the **ValueWhenBlank** text as a blank textbox.

- **ValueWhenBlankCssClass** (string) – When the TextBox is empty and this is assigned, the style sheet class name of the TextBox is assigned to this value. Use it to give the TextBox a different appearance when it is blank. This can help direct users to fields that they need to fill in. On the client-side, as focus is set to this field, the original style sheet class name will be restored to empty if the **PeterBlum.DES.Globals.Page.ValueWhenBlankMode** property is `RemoveText` or `RemoveBoth`. As focus leaves this field, when the field value is "", this class name is reassigned.

  Note: This property is only used when client-side scripting is available.

  Note: If the TextBox is invalid due to a validator assigned to it, the PeterBlum.DES.Globals.Page.ControlErrorCssClass property takes precedence over ValueWhenBlankCssClass. So a blank field with an error will show PeterBlum.DES.Globals.Page.ControlErrorCssClass when that property is set up.

  You can set up a global value for this control using the **DefaultValueWhenBlankCssClass** property within the **Global Settings Editor**. Once established, assign this property to "{DEFAULT}" to use that default.

  To merge your style sheet with the current style sheet class, put a plus ("+") character first. For example, "+MyClassName". Your style sheet class's attributes will override any attributes that it shares with the original class. The **DefaultValueWhenBlankCssClass** property can also start with "+" to have this effect.

  When this property is "", no style sheet class name is assigned for a blank field.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

ToolTip Properties

The Properties Editor shows these properties in the "ToolTip" category.

Note: The terms "Hint" and "ToolTip" both describe ways to provide documentation to the user. A Hint displays the message when focus enters the field and is best for data entry controls. A ToolTip displays the message when the mouse points to the control. It can be used on almost any type of control.

- **ToolTip** (string) – When assigned, a tooltip with this text is shown when the user points to the textbox. If you are using the **Hint** feature, it can be used as the hint when the **Hint** property is "". When using the "Enhanced ToolTips" feature, the browser's tooltip will be replaced by a PopupView. See the **Interactive Pages User's Guide**.

- **ToolTipLookupID** (string) – Gets the value for **ToolTip** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Hints. If no match is found OR this is blank, **ToolTip** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

- **ToolTipUsesPopupViewName** (string) – When using the "Enhanced ToolTips" feature, this determines which PopupView definition is used. For details on Enhanced ToolTips, see the **Interactive Pages User's Guide**.

  Specify the name from the PopupView definition or use the token "{DEFAULT}" to select the name from the global setting **DefaultToolTipPopupViewName**, which is set with the **Global Settings Editor**.

  A PopupView definition describes the name, style sheets, images, behaviors, and size of a PopupView. Use the **Global Settings** Editor to create and edit these PopupView definitions in the "PopupView definitions used by the HintManager" section.

  Tooltips are only converted to PopupViews when **HintManager.EnableToolTipsUsePopupViews** is `True`. (**HintManager** is accessed programmatically through **PeterBlum.DES.Globals.Page** and in the **PageManager** control.)

  Here are the predefined values: `LtYellow-Small`, `LtYellow-Medium`, `LtYellow-Large`, `ToolTip-Small`, `ToolTip-Medium`, and `ToolTip-Large`. All of these are light yellow. Their widths vary from 200px to 600px. Those named "ToolTip" have the callout feature disabled. Those named "LtYellow" have the callout feature enabled.

  It defaults to "`{DEFAULT}`".

  Note: When the name is unknown, it also uses the factory default. This allows the software to operate even if a PopupView definition is deleted or renamed.

  Note: When the HintManager.ToolTipsAsHints feature is enabled, anything other than "" or "{DEFAULT}" assigned to ToolTipUsesPopupViewName will prevent the ToolTip text from being assigned as a Hint. You must explicitly assign the Hint text if you want the tooltip and hint to share the same text.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Hint Properties

License Note: This feature requires a license for the Peter's Interactive Pages.

For an overview, see "Interactive Hints".

The Properties Editor shows these properties in the "Hint" category.

Note: The terms "Hint" and "ToolTip" both describe ways to provide documentation to the user. A Hint displays the message when focus enters the field and is best for data entry controls. A ToolTip displays the message when the mouse points to the control. It can be used on almost any type of control.

- **Hint** (string) – When using the Interactive Hints system, this is the text of the hint.

  When blank, if the TextBox is using its **ToolTip** property, the **ToolTip** is used as the text of the hint unless you set the **HintManager.ToolTipsAsHints** property to `False`.

  HTML tags are permitted. ENTER and LINEFEED characters are not. Use the token "{NEWLINE}" where you need a linefeed.

  When the hint is shown in the browser's status bar, HTML tags will automatically be stripped.

  It defaults to "".

- **HintLookupID** (string) – Gets the value for **Hint** through the String Lookup System. (See "The String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Hint. If no match is found OR this is blank, **Hint** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

- **HintHelp** (string) – When the Hint uses a PopupView, this provides data for use by the Help Button and other features on the PopupView. Its use depends on the **PopupView.HelpBehavior** property. (The PopupView is determined by the HintFormatter with its **PopupViewName** property.)

  The PopupView has an optional Help button. When setup, the user can click it to bring up additional information, such as a new page of help text.

  Here is how to use the **HintHelp** based on **PopupView.HelpBehavior**:

  - `None` - Do not show a Help Button. The **HintHelp** property is not used.
  - `ButtonAppends` - Add the text from **HintHelp** after the existing message. Use **PopupView.AppendHelpSeparator** to separate the two parts. When clicked, the Help button disappears and the message box is redrawn.
  - `ButtonReplaces` - Replace the text in the message with the **HintHelp**. When clicked, the Help button disappears and the message box is redrawn.
  - `Title` - The text appears in the header as the title. It replaces the **PopupView.HeaderText**. There is no Help Button. If **HintHelp** is blank, **PopupView.HeaderText** is used.
  - `Hyperlink` - Provide a Hyperlink. The Help Info text will appear in the "{0}" token of **PopupView.HyperlinkUrlForHelpButton**.

    For example, the **HyperlinkUrlForHelpButton** property may be "{0}" and this property is the complete URL "/helpfiles/helptopic1000.aspx".

    Another example uses the token for just a querystring parameter, like this: **HyperlinkUrlForHelpButton** = "/gethelp.aspx?topicid={0}" and this property contains the number of the ID.
  - `HyperlinkNewWindow` - Provide a Hyperlink that opens a new window. The **HintHelp** text will appear in the "{0}" token of **PopupView.HyperlinkUrlForHelpButton**.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- `ButtonRunsScript` - Runs the script supplied in **PopupView.ScriptForHelpButton**. The **HintHelp** text will replace the token "{0}" in that script.

This defaults to "".

- **HintHelpLookupID** (string) – Gets the value for **HintHelp** through the String Lookup System. (See "The String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Hint. If no match is found OR this is blank, **HintHelp** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

- **SharedHintFormatterName** (string) – Specify the name of the desired HintFormatter object found in **HintManager.SharedHintFormatters**. (HintManager is accessed programmatically through **PeterBlum.DES.Globals.Page** and in the **PageManager** control.) Alternatively, specify the name of a PopupView defined in the "PopupView definitions used by HintFormatters" of the **Global Settings Editor**.

  The `PeterBlum.DES.HintFormatter` class describes how the hint text will be displayed. It provides its name, display mode - on the page or in a PopupView, if it's also in the tooltip and/or status bar, and more.

  The **HintManager.SharedHintFormatters** property defines various ways to display a hint with `PeterBlum.DES.HintFormatter` objects. It lets you share a HintFormatter definition amongst controls on this page. It not only makes changes to the HintFormatter quick, but it also reduces the JavaScript output. If you want to create a HintFormatter specific to this control, set **SharedHintFormatterName** to "" and edit the properties of **LocalHintFormatter** (see below).

  If you specify the name of a PopupView and there is a definition with that name, a HintFormatter is automatically added to **HintManager.SharedHintFormatters** with its name matching the name of the PopupView. This is an easy way to work with PopupViews without the extra step of setting up HintFormatters. The HintFormatter defined will also show the hint as a tooltip but it will not show the hint in the status bar. If you need more control over the HintFormatter's properties, you must create the HintFormatter yourself.

  See the "Interactive Hints" section of the **Interactive Pages User's Guide** for details on the `PeterBlum.DES.HintFormatter` class and setting up **HintManager.SharedHintFormatters**.

  Use the token "{DEFAULT}" to get the name from **HintManager.DefaultSharedHintFormatterName**.

  It defaults to "`{DEFAULT}`".

- **LocalHintFormatter** (PeterBlum.DES.HintFormatter) – When none of the HintFormatter objects defined in **HintManager.SharedHintFormatters** is appropriate, use this property. (HintManager is accessed programmatically through **PeterBlum.DES.Globals.Page** and in the **PageManager** control.)

  The `PeterBlum.DES.HintFormatter` class describes how the hint text will be displayed. It provides its display mode - on the page or in a PopupView, if it's also in the tooltip and/or status bar, and more. See the "Interactive Hints" section of the **Interactive Pages User's Guide** for directions on using the `PeterBlum.DES.HintFormatter` class.

  You must set **SharedHintFormatterName** to "" for this to be used.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Tab Rules Properties

These properties support special forms of tabbing to another control. These features are only available on the client-side. When the browser does not support client-side scripts or **PeterBlum.DES.Globals.Page.JavaScriptEnabled** is `false`, these features have no effect.

The Properties Editor shows these properties in the "Tab Rules" category.

- **NextControlID** (string) – Sets up special tabbing to another control together with one or more of these properties: **TabAtMaxLength**, **TabOnEnterKey**, **TabByArrowKeys**, and **TabOnTheseKeys**. Specifies the ID to a control that will receive focus when one of the special tabbing features is detected.

  Use it when the next control is in the same or any parent naming container. If it's in another naming container, use **NextControl**.

  If assigned to an unknown control ID or one in an incorrect naming container, an exception will be thrown at runtime.

  When "", special tabbing is not set up. It defaults to "".

- **NextControl** (System.Web.UI.Control) – This is an alternative to **NextControlID**. It has the same features as **NextControlID**. It is assigned a reference to a control instead of an ID. As a result, it supports controls in any naming container. It must be assigned programmatically.

  When programmatically assigning properties to a TextBox control, if you have access to the next control object, it is better to assign it here than assign its ID to the **NextControlID** property because DES operates faster using **NextControl**.

- **PreviousControlID** (string) – Sets up special tabbing to another control together with one or more of these properties: **TabOnBackspace** and **TabByArrowKeys**. Specifies the ID to a control that will receive focus when one of the special tabbing features is detected.

  Use it when the Previous control is in the same or any parent naming container. If it's in another naming container, use **PreviousControl**.

  If assigned to an unknown control ID or one in an incorrect naming container, an exception will be thrown at runtime.

  When "", special tabbing is not set up. It defaults to "".

- **PreviousControl** (System.Web.UI.Control) – This is an alternative to **PreviousControlID**. It has the same features as **PreviousControlID**. It is assigned a reference to a control instead of an ID. As a result, it supports controls in any naming container. It must be assigned programmatically.

  When programmatically assigning properties to a TextBox control, if you have access to the Previous control object, it is better to assign it here than assign its ID to the **PreviousControlID** property because DES operates faster using **PreviousControl**.

- **TabAtMaxLength** (Boolean) – When **NextControlID** and **MaxLength** are set, it will automatically tab to the next control when the number of characters entered reaches **MaxLength**. It only works on a SingleLine textbox.

  When `true`, it is enabled but it will do nothing unless **NextControlID** and **MaxLength** are both set. It defaults to `true`.

- **TabByArrowKeys** (Boolean) – Determines if the arrow keys move focus to the **NextControlID** or **PreviousControlID** when the cursor reaches the text limit.

  Focus will move to **NextControlID** when the user types a right arrow at the end of the current text.

  Focus will move to **PreviousControlID** when the user types a left arrow at the start of the current text.

  It only works on a SingleLine textbox.

  It defaults to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **TabOnBackspace** (Boolean) – Determines if focus will move to the **PreviousControlID** when the user types a backspace into an empty textbox.

  It only works on a SingleLine textbox.

  It defaults to `false`.

- **TabOnEnterKey** (Boolean) – When **NextControl** is set and this is `true`, set focus to it when the ENTER key is pressed.

  Supported by Multiline fields as well as single line fields.

  It defaults to `false`.

  Be aware of these issues:

  - It prevents ENTER from clicking the default button on the page. However, if you use the **EnterSubmitsControlID** property, **EnterSubmitsControlID** overrides this property.
  - Multiline fields may support ENTER as a valid character. This prevents the ability to type ENTER into the text.

- **TabOnTheseKeys** (string) – When **NextControlID** is set, set focus to it when any character in this property is pressed.

  This feature is designed to let users type separator characters between textboxes such as the ")" after a textbox for the area code of a phone number.

  Use **TabOnEnterKey** to specify the ENTER key.

  When "", the feature is disabled.

  It defaults to "".

  The space character is legal along with any other character. Be very careful to select keys that you don't want entered because it will not enter any of these (unless client-side scripting is disabled).

  It never tabs when the textbox is empty to allow characters being entered from the previous textbox that uses autotabbing to be discarded. For example, if you have a phone number of three segments, the user may hit ")" to jump from the first textbox to the second. They also may type ")[space]". That space could be in TabOnTheseKeys for the second textbox and cause it to tab again. When the 2nd textbox is empty, the [space] is abandoned.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Client-Side Functions Properties

The Enhanced TextBox utilizes the client-side onkeypress and onkeydown events. These properties let you incorporate your own code into DES's onkeypress and onkeydown events.

The Properties Editor shows these properties in the "Tab To Next Control" category.

- **CustomKeyPressFunctionName** (string) – Use this property to insert your own client-side code into the onkeypress event. You can use this code to run your own custom commands and even filter out a character if desired.

  The onkeypress event is the best place to handle all of the keys that can appear in the textbox while the onkeydown event is often better for cursor movement keys. For onkeydown events, see **CustomKeyDownFunctionName**.

  Always use this instead of modifying the onkeypress event directly because the TextBox uses the onkeypress event and needs to return `true` or `false` to handle filtering.

  You must define a JavaScript function with three parameters and return a boolean value. Your function is called for each keystroke passed on the onkeypress event. The **CustomKeyPressFunctionName** property is assigned to the function name.

  The parameters are:

  - Fld - The object representing the textbox element.
  - Event - The event object. Use it to get additional information about the keystroke like modifier keys. You don't need to get the keycode from it due to the next parameter.
  - KeyCode - An integer containing the keycode of the key typed. Use this to determine what action to take.

  Return `true` if the keystroke should continue to be processed by the TextBox's code. Return `false` if the keystroke has been used and should not be used by any of the TextBox's code and it should be filtered out.

```
function FunctionName(pFld, pEvent, pKeyCode)
{
   // evaluate pKeyCode
   if (allow_the_KeyCode)
      return true;
   else
      return false;
}
```

  Your code needs to be declared on the page. See "Adding Your JavaScript to the Page".

  Define the name of the function in this property. If "", no function is defined.

  It defaults to "".

  ALERT: Many users make the mistake of assigning JavaScript code to this property. This will cause JavaScript errors. GOOD: "MyFunction". BAD: "MyFunction();" and "alert('stop it')".

  Note: JavaScript is case sensitive. Be sure the value of this property exactly matches the function definition.

- **CustomKeyDownFunctionName** (string) – Use this property to insert your own client-side code into the onkeydown event. You can use this code to run your own custom commands and even filter out a character if desired.

  The onkeypress event is the best place to handle all of the keys that can appear in the textbox while the onkeydown event is often better for cursor movement keys. For onkeypress events, see **CustomKeyPressFunctionName**.

  Always use this instead of modifying the onkeydown event directly because the TextBox uses the onkeypress event and needs to return `true` or `false` to handle filtering.

  You must define a JavaScript function with three parameters and return a boolean value. Your function is called for each keystroke passed on the onkeypress event. The **CustomKeyDownFunctionName** property is assigned to the function name.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

The parameters are:

- Fld - The object representing the textbox element.
- Event - The event object. Use it to get additional information about the keystroke like modifier keys. You don't need to get the keycode from it due to the next parameter.
- KeyCode - An integer containing the keycode of the key typed. Use this to determine what action to take.

Return `true` if the keystroke should continue to be processed by the TextBox's code. Return `false` if the keystroke has been used and should not be used by any of the TextBox's code and it should be filtered out.

```
function FunctionName(pFld, pEvent, pKeyCode)
{
   // evaluate pKeyCode
   if (allow_the_KeyCode)
      return true;
   else
      return false;
}
```

Your code needs to be declared on the page. See “Adding Your JavaScript to the Page”.

Define the name of the function in this property. If "", no function is defined.

It defaults to "".

ALERT: Many users make the mistake of assigning JavaScript code to this property. This will cause JavaScript errors. GOOD: “MyFunction”. BAD: “MyFunction();” and “alert(‘stop it’)”.

Note: JavaScript is case sensitive. Be sure the value of this property exactly matches the function definition.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

IntegerTextBox Control

The `PeterBlum.DES.IntegerTextBox` is a TextBox designed for integer data entry. It knows how to convert an integer into text and back. It has properties to determine if it supports negative numbers and the thousands separator. You can add a spinner control for the user to increment the integer.

Note: If you need a control to enter a number that is treated like a string, such as an account number, consider using the FilteredTextBox, with a filter that limits the text to digits.

Click on any of these topics to jump to them:

- Features
- Using the IntegerTextBox
  - Getting and Setting the Value of the TextBox
  - Formatting The Text
  - Adding A Spinner
  - Connecting Data To Other Fields On The Client-Side
  - When the TextBox is Empty
  - Validation on AutoPostBack
  - Interactive Hints
  - AutoComplete and "Smart Change System"
  - Other Behaviors
- Adding a IntegerTextBox
- Properties for the IntegerTextBox

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Features

Use demos here: http://www.peterblum.com/DES/DemoTextBoxes.aspx.

- It subclasses from `PeterBlum.DES.TextBox`, inheriting all of its qualities. See "Enhanced TextBox".
- Accepts positive and negative integers. There is a property to only permit non-negative values.
- Get and set the value of the textbox using an integer data type instead of a string, avoiding you having to write conversion code.
- Set a property to allow or prevent entry of negative numbers.
- It is culture sensitive, respecting the CultureInfo object defined on the **PeterBlum.DES.Globals.Page.CultureInfo** property.
- On the client-side, it filters keystrokes so users can only enter characters suitable to for integer entry.
- On the client-side, it reformats if needed as the user exits the textbox. For example, if the user enters "1000", it reformats to "1,000" (when showing the optional thousands separator.)
- When you attach a Validator to them, the Validator automatically configures itself to evaluate an integer with the other data entry rules specified on this control.
- Use a spinner control (up and down arrows to the right of the textbox) to increment the value.
- Use up and down arrow keys to increment the value.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the IntegerTextBox

Use demos here: http://www.peterblum.com/DES/DemoTextBoxes.aspx.

The `PeterBlum.DES.IntegerTextBox` is subclassed from `PeterBlum.DES.TextBox`, which is an enhanced version of the TextBox control supplied with ASP.NET. If you know how to use the ASP.NET TextBox, you already know how to use this control. See "Using the Enhanced TextBox Control".

Click on any of these topics to jump to them:

- Getting and Setting the Value of the TextBox
- Formatting The Text
- Adding A Spinner
- Connecting Data To Other Fields On The Client-Side

The following topics are inherited from DES's TextBox control:

- When the TextBox is Empty
- Validation on AutoPostBack
- Interactive Hints
- AutoComplete and "Smart Change System"
- Other Behaviors

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Getting and Setting the Value of the TextBox

When getting or setting the value, use the **IntegerValue** property instead of the **Text** property. These properties convert your integer or double into the localized text shown in the textbox.

Note: This control uses 32 bit integers to represent values. As a result, it has a range of -2,147,483,648 to 2,147,483,648. Values outside of this range are treated as errors. The DataTypeCheckValidator respects this and reports an error on values outside of the range.

If you are using DataBinding, consider these alternative properties: **IntegerBindable**, **IntegerOrZero**, and **IntegerNullable**.

By default, negatives are not permitted. If allow negative percentages, set **AllowNegatives** to `true`. The DataTypeCheckValidator will respect this setting and report an error when a negative number is found.

You can check for a blank textbox by checking the **IsEmpty** property for `true`. The IntegerTextBox also contain an invalid entry, such as an illegal characters or badly formatted entry. You can check for an invalid entry by checking the **IsValid** property for `false`, although a validator should avoid having to test for an invalid value.

When you need a server side event that is fired when the textbox's value has changed, use the **TextChanged** event.

Data Entry Validation

Consider validation a manditory part of data entry, whether it's an IntegerTextBox or just an ordinary textbox. It prevents illegal entries from getting into your database. Even if the IntegerTextBox enforces some rules for you, users may not have a browser that supports the client-side code for the IntegerTextBox, or its validators for that matter. Hackers often turn off javascript in their browser in hopes that your server side code doesn't protect against their illegal data.

Validation Guidelines:

- At minimum, validate the percentage number format with one of these validators.
    - When using the DES Validation Framework, assign a DataTypeCheckValidator to each IntegerTextBox. (See the **Validation User's Guide**.)
    - When using the Native Validation Framework, use DES's CompareValidator with the **Operator** property set to `DataTypeCheck`. (This validator is in the **PeterBlum.DES.NativeValidators** assembly. If you haven't done so, add this assembly to its own tab in the Visual Studio/VWD toolbox. See the **Installation Guide** for details.) See "Validation with the Native Validation Framework".
- Set up server side validation.
    - When using DES's validation framework, test **PeterBlum.DES.Globals.Page.IsValid** in your postback event handler methods. Only use the data if it is `true`.
    - When using the native validation framework, test **Page.IsValid** in your postback event handler. Only use the data if it is `true`.
- You can establish a numeric range to prevent selection outside the range using **MinValue** and **MaxValue**. Add the RangeValidator to report errors when the user types numbers outside the range. DES provides two RangeValidators, one for its own validation framework and the other for the native validation framework. Choose the correct one. *Do not use the original RangeValidator that comes with ASP.NET.*

  Alternatively, if you feel an out of range value can be reported under the error message from your DataTypeCheckValidator (or CompareValidator in the Native Validation Framework), set the **DataTypeCheckReportsRangeErrors** to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Formatting The Text

DES uses the System.Globalization.CultureInfo class to define numeric formatting rules. You establish the desired CultureInfo on the **PeterBlum.DES.Globals.Page.CultureInfo** property. Integers get the following values from **CultureInfo.NumberFormat**:

- Thousands separator from CultureInfo.NumberFormat.NumberGroupSeparator.
- Negative number format from CultureInfo.NumberFormat.NumberNegativePattern.

If you want to show thousands separators, set **ShowThousandsSeparator** to `true`.

Use the CalculationController (part of the DES: Client-side Toolkit) to perform math based on these textboxes. See the **Peter's Interactive Pages User's Guide**.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Adding A Spinner

A Spinner is a control with two arrows that appear to the right of the textbox. For example:

When the user clicks on an arrow, it increments or decrements the value. If they hold down the button, it repeatedly changes the value and after 5 cycles, its speed increases.

To use the Spinner, set **ShowSpinner** to `true`. Use the **SpinnerManager** object to change the URL to the arrow button images and the auto repeat speed. (**SpinnerManager** is a property of **PeterBlum.DES.Globals.Page** and the PageManager control.)

Note: Spinners are only supported on these browsers: IE Windows 5+, Netscape 7+, Mozilla 1.1+, FireFox, Opera 7+, and Safari.

If you have client-side code that shows or hides the textbox, call the function `DES_Refresh()` to tell DES to change the visibility of the spinners. *This is handled automatically when using the FieldStateController.*

When the spinner is shown, the control attempts to keep the spinner side-by-side with the textbox by enclosing the two elements in either a `<span>` or `<table>` tag, depending on the browser. If you want to remove the enclosure, set **ContainerMode** to `None`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Connecting Data To Other Fields On The Client-Side

Often users want to update other textboxes and labels when a value is changed in an IntegerTextBox. Use the CalculationController to perform math based on these textboxes. It can update a textbox or label for you. See the **Peter's Interactive Pages User's Guide**.

For other situations, write your own client-side JavaScript code. DES supplies a number of JavaScript functions to greatly simplify working with the PercentTextBoxes on the client-side. See "JavaScript Support Functions".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Adding a IntegerTextBox

These steps ask you to jump around the document using clicks on links. Adobe Reader offers a **Previous View** command to return to the link. Look for this in the Adobe Reader (shown v6.0)

1. Prepare the page for DES controls. See "Preparing a page for DES controls" in the **General Features Guide**. It covers issues like style sheets, AJAX, and localization.
2. Add an IntegerTextBox control to the page.

   Visual Studio and Visual Web Developer Design Mode Users

   Drag the IntegerTextBox control from the Toolbox onto your web form.

   Text Entry Users

   Add the control (inside the <form> area):

   ```
   <des:IntegerTextBox id="[YourControlID]" runat="server" />
   ```

   Programmatically creating the IntegerTextBox control

   - Identify the control which you will add the IntegerTextBox control to its **Controls** collection. Like all ASP.NET controls, the TextBox can be added to any control that supports child controls, like Panel, User Control, or TableCell. If you want to add it directly to the Page, first add a PlaceHolder at the desired location and use the PlaceHolder.
   - Create an instance of the IntegerTextBox control class. The constructor takes no parameters.
   - Assign the **ID** property.
   - Add the IntegerTextBox control to the **Controls** collection.

   In this example, the IntegerTextBox is created with an **ID** of "IntegerTextBox1". It is added to PlaceHolder1.

   [C#]

   ```
   PeterBlum.DES.IntegerTextBox vTextBox = new PeterBlum.DES.IntegerTextBox();
   vTextBox.ID = "IntegerTextBox1";
   PlaceHolder1.Controls.Add(vTextBox);
   ```

   *Note: The namespace for these controls is PeterBlum.DES. If you prefer, add a **using** clause to that namespace on your form.*

   [VB]

   ```
   Dim vTextBox As PeterBlum.DES.IntegerTextBox = _
      New PeterBlum.DES.IntegerTextBox()
   vTextBox.ID = "IntegerTextBox1"
   PlaceHolder1.Controls.Add(vTextBox)
   ```

   *Note: The namespace for these controls is PeterBlum.DES. If you prefer, add an **Imports** clause to that namespace on your form.*

   Guidelines for setting properties

   - Design mode users can use the Properties Editor or the Expanded Properties Editor. (See "Expanded Properties Editor" in the **General Features Guide**.) The SmartTag also offers some of the most important properties.
   - Text entry users should add the properties into the `<des:ControlClass>` tag in this format: `propertyname="value"`
   - When setting a property programmatically, have a reference to the control's object and set the property according to your language's rules.

3. Set the properties associated with the IntegerTextBox. See "Properties for the IntegerTextBox".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

4. Assign Validators to the TextBox.

Using DES Validation Framework

See the **Validation User's Guide** for details on these validators.

- Always add the DataTypeCheckValidator to block formatting errors.
- Add a RangeValidator when you are using the **MinValue** and **MaxValue** properties.
- If you have two IntegerTextBoxes where one needs to be greater, less than, equal or not equal to the other, add a CompareTwoFieldsValidator.
- If you have two IntegerTextBoxes where one must be greater or less than the other by a specific number, add a DifferenceValidator. Specify the number in the **DifferenceValue** property.
- Be sure that server side validation has been correctly setup.

Using Native Validation Framework

These validators are found in the **PeterBlum.DES.NativeValidators** assembly. Do not use the original CompareValidator and RangeValidator that come with ASP.NET because they don't handle the rich data entry formats of the PercentTextBox. See "Validation with the Native Validation Framework".

- Always add the CompareValidator with its **Operator** set to `DataTypeCheck` to block formatting errors.
- Add a RangeValidator when you are using the **MinValue** and **MaxValue** properties.
- If you have two IntegerTextBoxes where one needs to be greater, less than, equal or not equal to the other, add a CompareValidator. Set its **ControlToValidate** property to the start IntegerTextBox and **ControlToCompare** property to the end IntegerTextBox.
- If you have two IntegerTextBoxes where one must be greater or less than the other by a specific number, add the DifferenceValidator.
- Be sure that server side validation has been correctly setup.

5. Get and set the value using the **IntegerValue** property. When databinding, use **IntegerBindable**.

   Usually you will set the initial value in the `Page_Load()` method when **Page.IsPostBack** is `false`. You will get the submitted value in your post back event method, after checking that **PeterBlum.DES.Globals.Page.IsValid** (DES validation framework) or **Page.IsValid** (native validation framework) is `true`.

6. Here are some other considerations:

   - If you are using an AJAX system to update this control, set the **InAJAXUpdate** property to `true`. Also make sure the PageManager control or AJAXManager object has been setup for AJAX. See "Using these Controls With AJAX" in the **General Features Guide**. *Failure to follow these directions can result in incorrect behavior and javascript errors.*
   - This control does not preserve most of its properties in the ViewState, to limit its impact on the page. If you need to use the ViewState to retain the value of a property, see "The ViewState and Preserving Properties for PostBack" in the **General Features Guide**.
   - If you encounter errors, see the "Troubleshooting" section for extensive topics based on several years of tech support's experience with customers.
   - See also "Additional Topics for Using These Controls".

Use demos here: http://www.peterblum.com/DES/DemoTextBoxes.aspx.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties for the IntegerTextBox

Most of the properties are inherited from Enhanced TextBox (see "Properties for the Enhanced TextBox") and the ASP.NET TextBox (see "System.Web.UI.WebControls.TextBox Members").

Click on any of these topics to jump to them:

- Getting And Setting The Value Properties
- Editing Properties
- Formatting Properties
- Spinner Properties

The following topics are inherited from DES's TextBox control:

- Editing Properties
- Behavior Properties
- Appearance Properties
- AutoPostBack Properties
- Value When Blank Properties
- ToolTip Properties
- Hint Properties
- Tab Rules Properties
- Client-Side Functions Properties

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Getting And Setting The Value Properties

There are numerous ways to get and set the numeric value of this textbox. Choose the best one for your needs.

Some of these properties are hidden from the Properties Editor as they are only intended to be used programmatically. When shown, they appear under the "Data" Category.

- **IntegerValue** (integer) – Gets and sets the value for this control using a `System.Int32` value. You generally access this property programmatically. This property is preferred over using the **Text** property. However, you can get and set the value on the **Text** property so long as you handle the conversion between string and Int32.

  Usually you will set the initial value in the `Page_Load()` method when **Page.IsPostBack** is `false`. You will get the submitted value in your post back event method, after checking that **PeterBlum.DES.Globals.Page.IsValid** (when using a DataTypeCheckValidator) or its own **IsValid** property is `true`.

  Note: If you attempt to get a value from IntegerValue when the text is incorrectly formatted, you will get a System.FormatException thrown.

  You can set the textbox value to "" either by assigning **Text** = "" or **IntegerValue** = `System.Int32.MinValue` or call `SetEmpty()`.

- **IntegerValueOrZero** (integer) – An alternative to **IntegerValue** that returns a value even if the text is blank or not a valid number. When it's blank or an invalid entry, it returns a value of 0. This is a read-only property.

- **IntegerBindable** (object) – An alternative to **IntegerValue** designed for handling multiple types. It accepts either a byte, sbyte, int16, int32, SqlInt16, SqlInt32, `null`, string, or `System.DbNull.Value`.

  It returns an integer (System.Int32) or Sql32 depending on **BindableMode**.

  It is preferred when using DataBinding, due to its flexibility with data types.

  When evaluating a string, the format must represent an integer value respecting the culture formatting of **PeterBlum.DES.Globals.Page.CultureInfo**.

- **BindableMode** (enum PeterBlum.DES.BindMode) – Determines how the **IntegerBindable** property returns its value, as either an Integer (32 bit) or SqlInteger.

  - `Normal` – Returns an Integer when the textbox is valid or `null`/`nothing` when the textbox is blank or has an illegal value. This is the default value.
  - `SqlTypes` – Returns a `System.Data.SqlTypes.SqlInteger` when the textbox is valid or `System.DBNull.Value` when the textbox is blank or has an illegal value.
  - `TwoWay` – Use this when you are using `System.Web.UI.WebControls.SqlDataSource` and other DataSource web controls to implement two-way databinding. Returns an Integer when the textbox is valid or `null`/`nothing` when the textbox is blank or has an illegal value.

- **IntegerNullable** (integer) – *Requires ASP.NET 2 or higher.* Read only. An alternative to **IntegerValue** that accepts either an integer (System.Int32) or `null`/`nothing`. When using `null`/`nothing`, it clears the textbox.

- **IsValid** (Boolean) – Determines if the contents of the TextBox represent an integer. It returns `true` when it does represent an integer. It always returns `false` when the **Text** property is blank (after trimming). Consider using a DataTypeCheckValidator instead of this property. This is a read-only property.

- **IsEmpty** (Boolean) – Determines if the text is blank (after trimming). You should not attempt to get the field's value when this returns `true` because a blank textbox has no value. Consider adding a RequiredTextValidator to this control to prevent getting blank text fields. This is a read-only property.

- **IntegerValueOrZero** (integer) – An alternative to **IntegerValue** that returns a value even if the text is blank or not a valid number. When it's blank or an invalid entry, it returns a value of 0. This is a read-only property.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **MinValue** (string) – Set the minimum value of a range. It affects the spinners, up/down arrow key commands, and is automatically used by the RangeValidator when its own **Minimum** property is unassigned.

  While it's a string type, it must represent an integer or double value. This property is great for design mode and ASP.NET Declarative Syntax. When programming, it's easier to use **MinValueAsInteger** because they don't require conversion from your integer to a string.

- **MaxValue** (string) – Set the maximum value of a range. It affects the spinners, up/down arrow key commands, and is automatically used by the RangeValidator when its own **Maximum** property is unassigned.

  While it's a string type, it must represent an integer or double value. This property is great for design mode and ASP.NET Declarative Syntax. When programming, it's easier to use **MaxValueAsInteger** because they don't require conversion from your integer to a string.

- **MinValueAsInteger** (Integer) – Set the minimum value of a range. Alternative to **MinValue** that takes an integer. Must be set programmatically. It affects the spinners, up/down arrow key commands, and is automatically used by the RangeValidator when its own **Minimum** property is unassigned.

- **MaxValueAsInteger** (Integer) – Set the maximum value of a range. Alternative to **MaxValue** that takes an integer. Must be set programmatically. It affects the spinners, up/down arrow key commands, and is automatically used by the RangeValidator when its own **Maximum** property is unassigned.

- **AllowNegatives** (Boolean) – Determines if negative numbers are permitted. When `true`, they are permitted. It defaults to `true`. When `false`, keyboard filtering will not allow the minus ("-") character, and the DataTypeCheckValidator will report errors when negative values are entered. It defaults to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Editing Properties

These Properties Editor shows these properties in the category "Editing".

- **ReadOnly** (Boolean) – Determines if the textbox is editable or not. When **ReadOnly** is `true`, it is not editable. However, the focus can enter the textbox. You can still permit editing by using the arrow keys (**UpDnKeysIncrement** = `true`) and with the spinners (**ShowSpinners** = `true`) by setting **ReadOnlyAllowsEdits** to `true`.
- **ReadOnlyAllowsEdits** (Boolean) – Overrides the **ReadOnly** property to let spinners and arrow keys edit a read-only textbox. It defaults to `false`.

  If you don't want to support arrows, set **UpDnKeysIncrement** to `false`.

  Not used when **ReadOnly** is `false`.
- **UpDnKeysIncrement** (Boolean) – Determines if the up and down arrow keys increment and decrement the value.

  When `true`, the up and down arrow keys increment and decrement. When `false`, they do not.

  It defaults to `true`.
- **UseKeyboardFiltering** (Boolean) – When `true`, the browser filters out characters that are not supported by the data type. When `false`, it allows all keystrokes. It defaults to `true`. Keyboard filtering is automatically disabled on browsers that do not support DES's client-side code.

  The filter for IntegerTextBox allows only digits the culture specific decimal point character (**CultureInfo.NumberFormat.NumberDecimalSeparator**) and minus ("-"). Minus depends on the **AllowNegatives** property.
- **OnChangeFunctionName** (string) – Allows you to extend the client side **onchange** event fired when the user exits the `TextBox or invokes one of its commands. During the **onchange** event, the number is validated, reformatted, and passed along to other objects.

  You create a JavaScript function and assign its name to the **OnChangeFunctionName** property. You can give it any legal JavaScript name you want.

  ALERT: Many users make the mistake of assigning JavaScript code to this property. This will cause JavaScript errors. GOOD: "MyFunction". BAD: "MyFunction();" and "alert('stop it')".

  Note: JavaScript is case sensitive. Be sure the value of this property exactly matches the function definition.

  See also "Adding Your JavaScript to the Page".

Defining the JavaScript Function

Declare a JavaScript function in your page with three parameters:

- TextBoxID (string) - The element ID to the text box of the date text box control. It matches the **ClientID** property of the TextBox.
- Value (integer) - The resulting number in the field. It will be `null` if the number was invalid or the textbox was blank.
- Error (boolean) – When `true`, the number was invalid including a bad format or an illegal number. When `false`, there is no error or the textbox is blank.

The function does not return anything.

Here is an example function which assigns the number to another IntegerTextBox whose ID is ITB2:

```
function MyOnChangeFnc(pTBId, pValue, pError)
{
   if (!pError)
      DES_SetDTTBValue('ITB2', pValue, true);  // if pValue=null, ITB2 is blank
}
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **OnChangeFunctionAlways** (boolean) – Used when **OnChangeFunctionName** is defined. When **OnChangeFunctionAlways** is `false`, your function will only be called when there is a valid number. When **OnChangeFunctionAlways** is `true`, it will be called on every change. You can detect these cases:
    - Valid number: *Value* parameter is an integer and *Error* is `false`.
    - Invalid number: *Value* parameter is `null` and *Error* is `true`.
    - Blank textbox: *Value* parameter is `null` and *Error* is `false`.

  See "JavaScript Support Functions" for numerous functions that can assist in your coding efforts.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Formatting Properties

These properties determine what text is considered valid for an integer. In addition, the **PeterBlum.DES.Globals.Page.CultureInfo** determines many formatting rules.

The following properties are identified in the category "Formatting" under the Properties Editor:

- **ShowThousandsSeparator** (Boolean) – When `true`, the thousand separator is added into the textbox as it is reformatted. The thousands separator character is defined in **PeterBlum.DES.Globals.Page.CultureInfo**. When `false`, no thousand characters are shown. If the user had entered them, they would be removed. It defaults to `false`.

  Note: This TextBox always permits entry of the thousands separator regardless of this property.

- **FillLeadZeros** (Integer) – Provides additional formatting when converting an integer to text by adding lead zeros. When > 0, it adds enough lead zeroes to match the value of this property. For example, if this is 4, all values will have 4 digits. Any number that does not offer 4 digits gets lead zeros to fill it. When the user enters "34", this reformats to "0034".

  This is a formatting feature; it doesn't change the numeric value. If you want to maintain the textual representation with lead zeros, get and set the data value with the **Text** property.

  When 0, it is not used.

  It defaults to 0.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Spinner Properties

The Spinner is an extension to the IntegerTextBox. It provides a pair of arrow buttons that increment or decrement the value of the textbox when clicked. You can customize the button appearance and autorepeat speed with properties on **PeterBlum.DES.Globals.Page.SpinnerManager**. It respects the limits established with the **MinValue** and **MaxValue** properties.

The following properties are identified in the category "Spinner" under the Properties Editor:

- **ShowSpinner** (Boolean) – When `true`, the spinner control is shown. When `false`, it is not. It defaults to `false`.
- **IncrementValue** (Double) – The number to add or subtract to the current value. It supports decimal values. It defaults to 1.0.
- **ContainerMode** (enum PeterBlum.DES.ContainerMode) – When the spinner is shown, the HTML output looks like this:

  ```
  <input type='text' [attributes]/><table>[spinner buttons]<table/>
  ```

  When you assign absolute positioning ("gridlayout"), the style attribute used for absolute positioning would normally get assigned to the `<input>` tag, only affecting its position. If nothing else was done, you would see the textbox in the correct location while the buttons would be elsewhere on the page.

  In addition, these tags can wrap around, especially when placed in a `<p>` tag or a table cell that is too small. The spinners appear below the textbox when that happens.

  Use the **ContainerMode** property to add either a `<span>` or `<table>` tag around this control that limits wrap around. This property is only used when you are showing spinners. When using absolute positioning, the styles used for positioning are moved into the container tag so all of this control's tags are grouped under a common position.

  The enumerated type `PeterBlum.DES.ContainerMode` has these values:

  - `None` – Do not create a container. Use it when you want to create your own container or otherwise control the formatting. If you need to absolutely position this control, you must assign the styles for absolute positioning to your own container.
  - `Auto` – Creates either a `<span>` or `<table>` depending on the browser's support of these styles: `white-space:nowrap` (used by `<span>`) and `display:inline-block` (used by `<table>`). Internet Explorer 5+ and the Mozilla-based browsers support `white-space:nowrap` (as determined by the **PeterBlum.DES.Globals.Page.Browser.MakeWhiteSpaceNoWrap** property.) Opera 7+ and Safari support `display:inline-block` (as determined by the **PeterBlum.DES.Globals.Page.Browser.MakeBlockInlineValue** property.)

    This is the default.

    Note: A tag generates much smaller HTML than the <table> tag.
  - `Span` – Creates a `<span>` tag. It uses the style `white-space:nowrap` to prevent wrapping. Some browsers do not support this style and it will be omitted, leaving the control susceptible to wrapping on those browsers. (The **PeterBlum.DES.Globals.Page.Browser.MakeWhiteSpaceNoWrap** property determines support for this style.)

    The `<span>` tag will also be assigned the style `vertical-align:text-bottom` to keep all elements in a reasonable layout. Yet, some browsers do not make the alignments for this style as well as Internet Explorer and Mozilla-based browsers.
  - `Table` – Creates a `<table>` around the controls with each HTML element in its own table cell. While this is the most reliable, it has these limitations on browsers other than Internet Explorer for Windows, Mozilla, FireFox, and Netscape 6+:

    Some browsers do not support the `display:inline` or `display:inline-block` style required to position the table inside a row of text. A `<span>` tag will be generated instead.

    It uses the style `vertical-align:middle` to attempt to align all elements. Yet, some browsers do not make the alignments for this style as well as Internet Explorer and Mozilla-based browsers.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

It sets `padding` and `margin` styles to 0px on all sides of the cells. It sets `cellpadding` and `cellspacing` to 0 as well. These attempt to remove any shifting imposed by the table's margins. Yet, some browsers do not make the positioning as well as Internet Explorer and Mozilla-based browsers. You may notice misalignment with other elements in the same row.

Note: A tag generates much smaller HTML than the <table> tag.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

DecimalTextBox Control

The `PeterBlum.DES.DecimalTextBox` is a TextBox designed for decimal number data entry. It knows how to convert a `System.Double` type into text and back. It has properties to determine the number of decimal places, trialing zeroes, if it supports negative numbers and shows the thousands separator. You can add a spinner control for the user to increment the value.

Click on any of these topics to jump to them:

- Features
- Using the DecimalTextBox
  - Getting and Setting the Value of the TextBox
  - Formatting The Text
  - Adding A Spinner
  - Connecting Data To Other Fields On The Client-Side
  - When the TextBox is Empty
  - Validation on AutoPostBack
  - Interactive Hints
  - AutoComplete and “Smart Change System”
  - Other Behaviors
- Adding a DecimalTextBox
- Properties for the DecimalTextBox

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Features

Use demos here: http://www.peterblum.com/DES/DemoTextBoxes.aspx.

- It subclasses from `PeterBlum.DES.TextBox`, inheriting all of its qualities. See "Enhanced TextBox".
- Get and set the value of the textbox using an integer data type instead of a string, avoiding you having to write conversion code.
- Set a property to allow or prevent entry of negative numbers.
- It is culture sensitive, respecting the CultureInfo object defined on the **PeterBlum.DES.Globals.Page.CultureInfo** property.
- On the client-side, it filters keystrokes so users can only enter characters suitable to for integer entry.
- On the client-side, it reformats if needed as the user exits the textbox. For example, if the user enters "1000", it reformats to "1,000" (when showing the optional thousands separator.)
- When you attach a Validator to them, the Validator automatically configures itself to evaluate an integer with the other data entry rules specified on this control.
- Use a spinner control (up and down arrows to the right of the textbox) to increment the value.
- Use up and down arrow keys to increment the value.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the DecimalTextBox

Use demos here: http://www.peterblum.com/DES/DemoTextBoxes.aspx.

The `PeterBlum.DES.DecimalTextBox` is subclassed from `PeterBlum.DES.TextBox`, which is an enhanced version of the TextBox control supplied with ASP.NET. If you know how to use the ASP.NET TextBox, you already know how to use this control. See "Using the Enhanced TextBox Control".

Click on any of these topics to jump to them:

- Getting and Setting the Value of the TextBox
- Formatting The Text
- Adding A Spinner
- Connecting Data To Other Fields On The Client-Side
- When the TextBox is Empty
- Validation on AutoPostBack
- Interactive Hints
- AutoComplete and "Smart Change System"
- Other Behaviors

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Getting and Setting the Value of the TextBox

When getting or setting the value, use the **DoubleValue** or **DecimalValue** property instead of the **Text** property. These properties convert your double or decimal into the localized text shown in the textbox.

If you are using DataBinding, consider these alternative properties: **DoubleBindable** and **DoubleNullable**.

You can limit the number of decimal places with **MaxDecimalPlaces**.

By default, negatives are not permitted. If allow negative numbers, set **AllowNegatives** to `true`. The DataTypeCheckValidator will respect this setting and report an error when a negative number is found.

You can check for a blank textbox by checking the **IsEmpty** property for `true`. The DecimalTextBox also contain an invalid entry, such as an illegal characters or badly formatted entry. You can check for an invalid entry by checking the **IsValid** property for `false`, although a validator should avoid having to test for an invalid value.

When you need a server side event that is fired when the textbox's value has changed, use the **TextChanged** event.

Data Entry Validation

Consider validation a manditory part of data entry, whether it's a DecimalTextBox or just an ordinary textbox. It prevents illegal entries from getting into your database. Even if the DecimalTextBox enforces some rules for you, users may not have a browser that supports the client-side code for the DecimalTextBox, or its validators for that matter. Hackers often turn off javascript in their browser in hopes that your server side code doesn't protect against their illegal data.

Validation Guidelines:

- At minimum, validate the decimal number format with one of these validators.
    - When using the DES Validation Framework, assign a DataTypeCheckValidator to each DecimalTextBox. (See the **Validation User's Guide**.)
    - When using the Native Validation Framework, use DES's CompareValidator with the **Operator** property set to `DataTypeCheck`. (This validator is in the **PeterBlum.DES.NativeValidators** assembly. If you haven't done so, add this assembly to its own tab in the Visual Studio/VWD toolbox. See the **Installation Guide** for details.) See "Validation with the Native Validation Framework".
- Set up server side validation.
    - When using DES's validation framework, test **PeterBlum.DES.Globals.Page.IsValid** in your postback event handler methods. Only use the data if it is `true`.
    - When using the native validation framework, test **Page.IsValid** in your postback event handler. Only use the data if it is `true`.
- You can establish a numeric range to prevent selection outside the range using **MinValue** and **MaxValue**. Add the RangeValidator to report errors when the user types numbers outside the range. DES provides two RangeValidators, one for its own validation framework and the other for the native validation framework. Choose the correct one. *Do not use the original RangeValidator that comes with ASP.NET.*

  Alternatively, if you feel an out of range value can be reported under the error message from your DataTypeCheckValidator (or CompareValidator in the Native Validation Framework), set the **DataTypeCheckReportsRangeErrors** to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Formatting The Text

DES uses the System.Globalization.CultureInfo class to define most numeric formatting rules. You establish the desired CultureInfo on the **PeterBlum.DES.Globals.Page.CultureInfo** property. Decimal values use the following properties from **CultureInfo.NumberFormat**:

- Decimal place character from CultureInfo.NumberFormat.NumberDecimalSeparator.
- Thousands separator from CultureInfo.NumberFormat.NumberGroupSeparator.
- Negative number format from CultureInfo.NumberFormat.NumberNegativePattern.

Other Formatting Rules

You can add trailing zeroes to the decimal places with **TrailingZeroDecimalPlaces**.

To show thousands separators, set **ShowThousandsSeparator** to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Adding A Spinner

A Spinner is a control with two arrows that appear to the right of the textbox. For example:

When the user clicks on an arrow, it increments or decrements the value. If they hold down the button, it repeatedly changes the value and after 5 cycles, its speed increases.

To use the Spinner, set **ShowSpinner** to `true`. Use the **SpinnerManager** object to change the URL to the arrow button images and the auto repeat speed. (**SpinnerManager** is a property of **PeterBlum.DES.Globals.Page** and the PageManager control.)

Note: Spinners are only supported on these browsers: IE Windows 5+, Netscape 7+, Mozilla 1.1+, FireFox, Opera 7+, and Safari.

If you have client-side code that shows or hides the textbox, call the function `DES_Refresh()` to tell DES to change the visibility of the spinners. *This is handled automatically when using the FieldStateController.*

When the spinner is shown, the control attempts to keep the spinner side-by-side with the textbox by enclosing the two elements in either a `<span>` or `<table>` tag, depending on the browser. If you want to remove the enclosure, set **ContainerMode** to `None`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Connecting Data To Other Fields On The Client-Side

Often users want to update other textboxes and labels when a value is changed in a DecimalTextBox. Use the CalculationController to perform math based on these textboxes. It can update a textbox or label for you. See the **Peter's Interactive Pages User's Guide**.

For other situations, write your own client-side JavaScript code. DES supplies a number of JavaScript functions to greatly simplify working with the DecimalTextBoxes on the client-side. See "JavaScript Support Functions".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Adding a DecimalTextBox

These steps ask you to jump around the document using clicks on links. Adobe Reader offers a **Previous View** command to return to the link. Look for this in the Adobe Reader (shown v6.0)

1. Prepare the page for DES controls. See "Preparing a page for DES controls" in the **General Features Guide**. It covers issues like style sheets, AJAX, and localization.
2. Add a DecimalTextBox control to the page.

   Visual Studio and Visual Web Developer Design Mode Users

   Drag the DecimalTextBox control from the Toolbox onto your web form.

   Text Entry Users

   Add the control (inside the <form> area):

   ```
   <des:DecimalTextBox id="[YourControlID]" runat="server" />
   ```

   Programmatically creating the DecimalTextBox control

   - Identify the control which you will add the DecimalTextBox control to its **Controls** collection. Like all ASP.NET controls, the TextBox can be added to any control that supports child controls, like Panel, User Control, or TableCell. If you want to add it directly to the Page, first add a PlaceHolder at the desired location and use the PlaceHolder.
   - Create an instance of the DecimalTextBox control class. The constructor takes no parameters.
   - Assign the **ID** property.
   - Add the DecimalTextBox control to the **Controls** collection.

   In this example, the DecimalTextBox is created with an **ID** of "DecimalTextBox1". It is added to PlaceHolder1.

   [C#]

   ```
   PeterBlum.DES.DecimalTextBox vTextBox = new PeterBlum.DES.DecimalTextBox();
   vTextBox.ID = "DecimalTextBox1";
   PlaceHolder1.Controls.Add(vTextBox);
   ```

   *Note: The namespace for these controls is PeterBlum.DES. If you prefer, add a **using** clause to that namespace on your form.*

   [VB]

   ```
   Dim vTextBox As PeterBlum.DES.DecimalTextBox =  _
      New PeterBlum.DES.DecimalTextBox()
   vTextBox.ID = "DecimalTextBox1"
   PlaceHolder1.Controls.Add(vTextBox)
   ```

   *Note: The namespace for these controls is PeterBlum.DES. If you prefer, add an **Imports** clause to that namespace on your form.*

   Guidelines for setting properties

   - Design mode users can use the Properties Editor or the Expanded Properties Editor. (See "Expanded Properties Editor" in the **General Features Guide**.) The SmartTag also offers some of the most important properties.
   - Text entry users should add the properties into the `<des:ControlClass>` tag in this format: `propertyname="value"`
   - When setting a property programmatically, have a reference to the control's object and set the property according to your language's rules.

3. Set the properties associated with the DecimalTextBox.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

4. Assign Validators to the TextBox.

Using DES Validation Framework

See the **Validation User's Guide** for details on these validators.

- Always add the DataTypeCheckValidator to block formatting errors.
- Add a RangeValidator if you are using the **MinValue** and **MaxValue** properties.
- If you have two DecimalTextBoxes where one needs to be greater, less than, equal or not equal to the other, add a CompareTwoFieldsValidator.
- If you have two DecimalTextBoxes where one must be greater or less than the other by a specific number, add a DifferenceValidator. Specify the number in the **DifferenceValue** property.
- Be sure that server side validation has been correctly set up.

Using Native Validation Framework

These validators are found in the **PeterBlum.DES.NativeValidators** assembly. Do not use the original CompareValidator and RangeValidator that come with ASP.NET because they don't handle the rich data entry formats of the DecimalTextBox. See "Validation with the Native Validation Framework".

- Always add the CompareValidator with its **Operator** set to `DataTypeCheck` to block formatting errors.
- Add a RangeValidator when you are using the **MinValue** and **MaxValue** properties.
- If you have two DecimalTextBoxes where one needs to be greater, less than, equal or not equal to the other, add a CompareValidator. Set its **ControlToValidate** property to the start DecimalTextBox and **ControlToCompare** property to the end DecimalTextBox.
- If you have two DecimalTextBoxes where one must be greater or less than the other by a specific number, add the DifferenceValidator.
- Be sure that server side validation has been correctly set up.

5. Get and set the value using the **DoubleValue** or **DecimalValue** property. When databinding, use **DoubleBindable**.

   Usually you will set the initial value in the `Page_Load()` method when **Page.IsPostBack** is `false`. You will get the submitted value in your post back event method, after checking that **PeterBlum.DES.Globals.Page.IsValid** (DES validation framework) or **Page.IsValid** (native validation framework) is `true`.

6. Here are some other considerations:

   - If you are using an AJAX system to update this control, set the **InAJAXUpdate** property to `true`. Also make sure the PageManager control or AJAXManager object has been setup for AJAX. See "Using these Controls With AJAX" in the **General Features Guide**. *Failure to follow these directions can result in incorrect behavior and javascript errors.*
   - This control does not preserve most of its properties in the ViewState, to limit its impact on the page. If you need to use the ViewState to retain the value of a property, see "The ViewState and Preserving Properties for PostBack" in the **General Features Guide**.
   - If you encounter errors, see the "Troubleshooting" section for extensive topics based on several years of tech support's experience with customers.
   - See also "Additional Topics for Using These Controls".

Use demos here: http://www.peterblum.com/DES/DemoTextBoxes.aspx.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties for the DecimalTextBox

Most of the properties are inherited from `PeterBlum.DES.TextBox` (see "Properties for the Enhanced TextBox") and the ASP.NET TextBox (see "System.Web.UI.WebControls.TextBox Members").

Click on any of these topics to jump to them:

- Getting And Setting The Value Properties
- Editing Properties
- Formatting Properties
- Spinners Properties

The following topics are inherited from DES's TextBox control:

- Editing Properties
- Behavior Properties
- Appearance Properties
- AutoPostBack Properties
- Value When Blank Properties
- ToolTip Properties
- Hint Properties
- Tab Rules Properties
- Client-Side Functions Properties

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Getting And Setting The Value Properties

There are numerous ways to get and set the numeric value of this textbox. Choose the best one for your needs.

Some of these properties are hidden from the Properties Editor as they are only intended to be used programmatically. When shown, they appear under the "Data" Category.

- **DoubleValue** (double) – Gets and sets the value for this control using a `System.Double` value. You generally access this property programmatically. This property is preferred over using the **Text** property. However, you can get and set the value on the **Text** property so long as you handle the conversion between string and Double.

  Usually you will set the initial value in the `Page_Load()` method when **Page.IsPostBack** is `false`. You will get the submitted value in your post back event method, after checking that **PeterBlum.DES.Globals.Page.IsValid** (when using a DataTypeCheckValidator) or its own **IsValid** property is `true`.

  Note: If you attempt to get a value from DoubleValue when the text is incorrectly formatted, you will get a System.FormatException thrown.

  You can set the textbox value to "" either by assigning **Text** = "" or **DoubleValue** = `System.Double.MinValue`.

  Note: While .net handles larger values with its System.Decimal type, on the client-side JavaScript is limited to something close to the range of the System.Double type.

- **DecimalValue** (decimal) – Gets and sets the value for this control using a `System.Decimal` value. *See also DoubleValue, IntegerValue, and other properties in this section as alternatives.* You generally access this property programmatically. This property is preferred over using the **Text** property. However, you can get and set the value on the **Text** property so long as you handle the conversion between string and Decimal.

  Usually you will set the initial value in the `Page_Load()` method when **Page.IsPostBack** is `false`. You will get the submitted value in your post back event method, after checking that **PeterBlum.DES.Globals.Page.IsValid** (when using a DataTypeCheckValidator) or its own **IsValid** property is `true`.

  Note: If you attempt to get a value from DecimalValue when the text is incorrectly formatted, you will get a System.FormatException thrown.

  You can set the textbox value to "" either by assigning **Text** = "" or **DecimalValue** = `System.Decimal.MinValue` or call `SetEmpty()`.

  Note: While .net handles larger values with its System.Decimal type, on the client-side JavaScript is limited to something close to the range of the System.Double type.

- **DoubleBindable** (object) – An alternative to **DoubleValue** designed for handling multiple types. It accepts either a decimal, double, single, int32, SqlDecimal, SqlDouble, SqlInt32, `null`, string, or `System.DbNull.Value`.

  It returns a Double or SqlDouble depending on **BindableMode**.

  It is preferred when using DataBinding, due to its flexibility with data types.

  When evaluating a string, the format must represent a decimal or integer value respecting the culture formatting of **PeterBlum.DES.Globals.Page.CultureInfo**.

- **BindableMode** (enum PeterBlum.DES.BindMode) – Determines how the **DoubleBindable** properties returns its value, as either a native type (32 bit integer or double) or a `System.Data.SqlTypes` type (SqlInteger or SqlDouble).

  - `Normal` – Returns a double when the textbox is valid or `null`/`nothing` when the textbox is blank or has an illegal value. This is the default value.
  - `SqlTypes` – Returns a SqlDouble when the textbox is valid or `System.DBNull.Value` when the textbox is blank or has an illegal value.
  - `TwoWay` – Use this when you are using `System.Web.UI.WebControls.SqlDataSource` and other DataSource web controls to implement two-way databinding. Returns a double when the textbox is valid or `null`/`nothing` when the textbox is blank or has an illegal value.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **DoubleNullable** (double) – *Requires ASP.NET 2 or higher.* Read only. An alternative to **DoubleValue** that accepts either a double or `null`/`nothing`. When using `null`/`nothing`, it clears the textbox.
- **DoubleValueOrZero** (double) – An alternative to **DoubleValue** that returns a value even if the text is blank or not a valid number. When it's blank or an invalid entry, it returns a value of 0.0. This is a read-only property.
- **IsValid** (Boolean) – Determines if the contents of the TextBox represent a number. It returns `true` when it does represent a number. It always returns `false` when the **Text** property is blank (after trimming). Consider using a DataTypeCheckValidator instead of this property. This is a read only property.
- **IsEmpty** (Boolean) – Determines if the text is blank (after trimming). You should not attempt to get the field's value when this returns `true` because a blank textbox has no value. Consider adding a RequiredTextValidator to this control to prevent getting blank text fields. This is a read only property.
- **MinValue** (string) – Set the minimum value of a range. It affects the spinners, up/down arrow key commands, and is automatically used by the RangeValidator when its own **Minimum** property is unassigned.

  While it's a string type, it must represent a double value. This property is great for design mode and ASP.NET Declarative Syntax. When programming, it's easier to use **MinValueAsDouble** because they don't require conversion from your double to a string.
- **MaxValue** (string) – Set the maximum value of a range. It affects the spinners, up/down arrow key commands, and is automatically used by the RangeValidator when its own **Maximum** property is unassigned.

  While it's a string type, it must represent a double value. This property is great for design mode and ASP.NET Declarative Syntax. When programming, it's easier to use **MaxValueAsDouble** because they don't require conversion from your double to a string.
- **MinValueAsDouble** (double) – Set the minimum value of a range. Alternative to **MinValue** that takes a double. Must be set programmatically. It affects the spinners, up/down arrow key commands, and is automatically used by the RangeValidator when its own **Minimum** property is unassigned.
- **MaxValueAsDouble** (double) – Set the maximum value of a range. Alternative to **MaxValue** that takes a double. Must be set programmatically. It affects the spinners, up/down arrow key commands, and is automatically used by the RangeValidator when its own **Maximum** property is unassigned.
- **AllowNegatives** (Boolean) – Determines if negative numbers are permitted. When `true`, they are permitted. It defaults to `true`. When `false`, keyboard filtering will not allow the minus ("-") character, and the DataTypeCheckValidator will report errors when negative values are entered. It defaults to `false`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Editing Properties

The Properties Editor shows these properties in the category "Editing".

- **ReadOnly** (Boolean) – Determines if the textbox is editable or not. When **ReadOnly** is `true`, it is not editable. However, the focus can enter the textbox. You can still permit editing by using the arrow keys (**UpDnKeysIncrement** = `true`) and with the spinners (**ShowSpinners** = `true`) by setting **ReadOnlyAllowsEdits** to `true`.
- **ReadOnlyAllowsEdits** (Boolean) – Overrides the **ReadOnly** property to let spinners and arrow keys edit a read-only textbox. It defaults to `false`.

  If you don't want to support arrows, set **UpDnKeysIncrement** to `false`.
- **UpDnKeysIncrement** (Boolean) – Determines if the up and down arrow keys increment and decrement the value.

  When `true`, the up and down arrow keys increment and decrement. When `false`, they do not.

  It defaults to `true`.
- **AcceptPeriodAsDecimalSeparator** (Boolean) – Many cultures do not use the period as the decimal separator. It makes numeric entry from the numeric keypad more difficult, because it features a period key. When setup, both the culture's decimal separator and a period are allowed as the decimal separator. The textbox will convert the period to the decimal character.

  Does not apply for cultures that already use a period for the decimal separator. Some cultures use a period as a thousands separator. In those cases, the parser will only consider periods the decimal separator when there is only one period character and the culture's decimal separator is not found.

  When `true`, the feature is used so long as the current culture's decimal separator is not already a period.

  When `false`, the feature is not used.

  It defaults to `false`.
- **UseKeyboardFiltering** (Boolean) – When `true`, the browser filters out characters that are not supported by the data type. When `false`, it allows all keystrokes. It defaults to `true`. Keyboard filtering is automatically disabled on browsers that do not support DES's client-side code.

  The filter for DecimalTextBox allows only digits the culture specific decimal point character (**CultureInfo.NumberFormat.NumberDecimalSeparator**) and minus ("-"). Minus depends on the **AllowNegatives** property.
- **OnChangeFunctionName** (string) – Allows you to extend the client side **onchange** event fired when the user exits the DecimalTextBox or invokes one of its commands. During the **onchange** event, the number is validated, reformatted, and passed along to other objects.

  You create a JavaScript function and assign its name to the **OnChangeFunctionName** property. You can give it any legal JavaScript name you want.

  ALERT: Many users make the mistake of assigning JavaScript code to this property. This will cause JavaScript errors. GOOD: "MyFunction". BAD: "MyFunction();" and "alert('stop it')".

  Note: JavaScript is case sensitive. Be sure the value of this property exactly matches the function definition.

  See also "Adding Your JavaScript to the Page".

Defining the JavaScript Function

Declare a JavaScript function in your page with three parameters:

- TextBoxID (string) - The element ID to the text box of the date text box control. It matches the **ClientID** property of the TextBox.
- Value (floating point) - The resulting number in the field. It will be `null` if the number was invalid or the textbox was blank.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- Error (boolean) – When `true`, the number was invalid including a bad format or an illegal number. When `false`, there is no error or the textbox is blank.

The function does not return anything.

Here is an example function which assigns the number to another DecimalTextBox whose ID is DTB2:

```
function MyOnChangeFnc(pTBId, pValue, pError)
{
   if (!pError)
      DES_SetDTTBValue('DTB2', pValue, true);  // if pValue=null, DTB2 is blank
}
```

- **OnChangeFunctionAlways** (boolean) – Used when **OnChangeFunctionName** is defined. When **OnChangeFunctionAlways** is `false`, your function will only be called when there is a valid number. When **OnChangeFunctionAlways** is `true`, it will be called on every change. You can detect these cases:
  - Valid number: *Value* parameter is an integer and *Error* is `false`.
  - Invalid number: *Value* parameter is `null` and *Error* is `true`.
  - Blank textbox: *Value* parameter is `null` and *Error* is `false`.

See "JavaScript Support Functions" for numerous functions that can assist in your coding efforts.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Formatting Properties

These properties determine what text is considered valid for a decimal number. In addition, the **PeterBlum.DES.Globals.Page.CultureInfo** determines many formatting rules.

- **ShowThousandsSeparator** (Boolean) – When `true`, the thousand separator is added into the textbox as it is reformatted. The thousands separator character is defined in **PeterBlum.DES.Globals.Page.CultureInfo**. When `false`, no thousand characters are shown. If the user had entered them, they would be removed. It defaults to `false`.

  Note: This TextBox always permits entry of the thousands separator regardless of this property.

- **TrailingZeroDecimalPlaces** (Integer) – Determines how many trailing decimal places should appear when reformatting a double value to a string.

  When the number of digits after the decimal point is less than this number, trailing zeros are added.

  When this value is 0, the value will be represented as a whole number with no decimal point when there are no non-zero digits after the decimal point. For example, "1.0" will be represented as "1".

  When this value is -1, no change is made to the formatting; this property is not used.

  It defaults to -1.

- **MaxDecimalPlaces** (Integer) – Determines the maximum number of decimal places allowed. When the user enters more, it is an error which the DataTypeCheckValidator will detect.

  Set to 0 to ignore this property. It defaults to 0.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Spinners Properties

The Spinner is an extension to the DecimalTextBox. It provides a pair of arrow buttons that increment or decrement the value of the textbox when clicked. You can customize the button appearance and autorepeat speed with properties on **PeterBlum.DES.Globals.Page.SpinnerManager**. It respects the limits established with the **MinValue** and **MaxValue** properties.

The following properties are identified in the category "Spinner" under the Properties Editor:

- **ShowSpinner** (Boolean) – When `true`, the spinner control is shown. When `false`, it is not. It defaults to `false`.
- **IncrementValue** (Double) – The number to add or subtract to the current value. It supports decimal values. It defaults to 1.0. If you use a decimal number, also set **MaxDecimalPlaces** to enough digits to hold the decimal portion. For example, if this is 0.25, set **MaxDecimalPlaces** to a minimum of 2.
- **ContainerMode** (enum PeterBlum.DES.ContainerMode) – When the spinner is shown, the HTML output looks like this:

  ```
  <input type='text' [attributes]/><table>[spinner buttons]<table/>
  ```

  When you assign absolute positioning ("gridlayout"), the style attribute used for absolute positioning would normally get assigned to the `<input>` tag, only affecting its position. If nothing else was done, you would see the textbox in the correct location while the buttons would be elsewhere on the page.

  In addition, these tags can wrap around, especially when placed in a `<p>` tag or a table cell that is too small. The spinners appear below the textbox when that happens.

  Use the **ContainerMode** property to add either a `<span>` or `<table>` tag around this control that limits wrap around. This property is only used when you are showing spinners. When using absolute positioning, the styles used for positioning are moved into the container tag so all of this control's tags are grouped under a common position.

  The enumerated type `PeterBlum.DES.ContainerMode` has these values:

  - `None` – Do not create a container. Use it when you want to create your own container or otherwise control the formatting. If you need to absolutely position this control, you must assign the styles for absolute positioning to your own container.
  - `Auto` – Creates either a `<span>` or `<table>` depending on the browser's support of these styles: `white-space:nowrap` (used by `<span>`) and `display:inline-block` (used by `<table>`). Internet Explorer 5+ and the Mozilla-based browsers support `white-space:nowrap` (as determined by the **PeterBlum.DES.Globals.Page.Browser.MakeWhiteSpaceNoWrap** property.) Opera 7+ and Safari support `display:inline-block` (as determined by the **PeterBlum.DES.Globals.Page.Browser.MakeBlockInlineValue** property.)

    This is the default.

    Note: A tag generates much smaller HTML than the <table> tag.
  - `Span` – Creates a `<span>` tag. It uses the style `white-space:nowrap` to prevent wrapping. Some browsers do not support this style and it will be omitted, leaving the control susceptible to wrapping on those browsers. (The **PeterBlum.DES.Globals.Page.Browser.MakeWhiteSpaceNoWrap** property determines support for this style.)

    The `<span>` tag will also be assigned the style `vertical-align:text-bottom` to keep all elements in a reasonable layout. Yet, some browsers do not make the alignments for this style as well as Internet Explorer and Mozilla-based browsers.
  - `Table` – Creates a `<table>` around the controls with each HTML element in its own table cell. While this is the most reliable, it has these limitations on browsers other than Internet Explorer for Windows, Mozilla, FireFox, and Netscape 6+:

    Some browsers do not support the `display:inline` or `display:inline-block` style required to position the table inside a row of text. A `<span>` tag will be generated instead.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

It uses the style `vertical-align:middle` to attempt to align all elements. Yet, some browsers do not make the alignments for this style as well as Internet Explorer and Mozilla-based browsers.

It sets `padding` and `margin` styles to 0px on all sides of the cells. It sets `cellpadding` and `cellspacing` to 0 as well. These attempt to remove any shifting imposed by the table's margins. Yet, some browsers do not make the positioning as well as Internet Explorer and Mozilla-based browsers. You may notice misalignment with other elements in the same row.

Note: A tag generates much smaller HTML than the <table> tag.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

CurrencyTextBox Control

The `PeterBlum.DES.CurrencyTextBox` is a TextBox designed for currency number data entry. A currency number is really just a decimal number that uses specialized rules for its formatting including currency symbol and how to formats negative values. The CurrencyTextBox knows how to convert a `System.Double` type into text and back. It has properties to determine how to handle decimal places, show the currency symbol, support for negative numbers and show the thousands separator. You can add a spinner control for the user to increment the value.

Click on any of these topics to jump to them:

- Features
- Using the CurrencyTextBox
  - Getting and Setting the Value of the TextBox
  - Formatting The Text
  - Adding A Spinner
  - Connecting Data To Other Fields On The Client-Side
  - When the TextBox is Empty
  - Validation on AutoPostBack
  - Interactive Hints
  - AutoComplete and “Smart Change System”
  - Other Behaviors
- Adding a CurrencyTextBox
- Properties for the CurrencyTextBox

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Features

Use demos here: http://www.peterblum.com/DES/DemoTextBoxes.aspx.

- It subclasses from `PeterBlum.DES.TextBox`, inheriting all of its qualities. See "Enhanced TextBox".
- Get and set the value of the textbox using a double or decimal data type instead of a string, avoiding you having to write conversion code.
- Set a property to allow or prevent entry of negative numbers.
- It optionally can show the currency symbol within the text.
- It is culture sensitive, respecting the CultureInfo object defined on the **PeterBlum.DES.Globals.Page.CultureInfo** property.
- On the client-side, it filters keystrokes so users can only enter characters suitable to for decimal entry.
- On the client-side, it reformats if needed as the user exits the textbox. For example, if the user enters "1000", it reformats to "$1,000.00" (when showing the optional thousands separator.)
- When you attach a Validator to them, the Validator automatically configures itself to evaluate a currency value with the other data entry rules specified on this control.
- Use a spinner control (up and down arrows to the right of the textbox) to increment the value.
- Use up and down arrow keys to increment the value.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the CurrencyTextBox

Use demos here: http://www.peterblum.com/DES/DemoTextBoxes.aspx.

The `PeterBlum.DES.CurrencyTextBox` is subclassed from `PeterBlum.DES.TextBox`, which is an enhanced version of the TextBox control supplied with ASP.NET. If you know how to use the ASP.NET TextBox, you already know how to use this control. See "Using the Enhanced TextBox Control".

Click on any of these topics to jump to them:

- Getting and Setting the Value of the TextBox
- Formatting The Text
- Adding A Spinner
- Connecting Data To Other Fields On The Client-Side

The following topics are inherited from DES's TextBox control:

- When the TextBox is Empty
- Validation on AutoPostBack
- Interactive Hints
- AutoComplete and "Smart Change System"
- Other Behaviors

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Getting and Setting the Value of the TextBox

When getting or setting the value, use the **DoubleValue**, or **DecimalValue** property instead of the **Text** property. These properties convert your integer or double into the localized text shown in the textbox.

If you are using DataBinding, consider these alternative properties: **DoubleBindable** and **DoubleNullable**.

If this field permits more decimal digits than the culture allows, set **AllowExtraDecimalDigits** to `true`.

By default, negatives are not permitted. If allow negative Currency, set **AllowNegatives** to `true`. The DataTypeCheckValidator will respect this setting and report an error when a negative number is found.

You can check for a blank textbox by checking the **IsEmpty** property for `true`. The CurrencyTextBox also contain an invalid entry, such as an illegal characters or badly formatted entry. You can check for an invalid entry by checking the **IsValid** property for `false`, although a validator should avoid having to test for an invalid value.

When you need a server side event that is fired when the textbox's value has changed, use the **TextChanged** event.

Data Entry Validation

Consider validation a manditory part of data entry, whether it's a CurrencyTextBox or just an ordinary textbox. It prevents illegal entries from getting into your database. Even if the CurrencyTextBox enforces some rules for you, users may not have a browser that supports the client-side code for the CurrencyTextBox, or its validators for that matter. Hackers often turn off javascript in their browser in hopes that your server side code doesn't protect against their illegal data.

Validation Guidelines:

- At minimum, validate the currency number format with one of these validators.
    - When using the DES Validation Framework, assign a DataTypeCheckValidator to each CurrencyTextBox. (See the **Validation User's Guide**.)
    - When using the Native Validation Framework, use DES's CompareValidator with the **Operator** property set to `DataTypeCheck`. (This validator is in the **PeterBlum.DES.NativeValidators** assembly. If you haven't done so, add this assembly to its own tab in the Visual Studio/VWD toolbox. See the **Installation Guide** for details.) See "Validation with the Native Validation Framework".
- Set up server side validation.
    - When using DES's validation framework, test **PeterBlum.DES.Globals.Page.IsValid** in your postback event handler methods. Only use the data if it is `true`.
    - When using the native validation framework, test **Page.IsValid** in your postback event handler. Only use the data if it is `true`.
- You can establish a numeric range to prevent selection outside the range using **MinValue** and **MaxValue**. Add the RangeValidator to report errors when the user types numbers outside the range. DES provides two RangeValidators, one for its own validation framework and the other for the native validation framework. Choose the correct one. *Do not use the original RangeValidator that comes with ASP.NET.*

  Alternatively, if you feel an out of range value can be reported under the error message from your DataTypeCheckValidator (or CompareValidator in the Native Validation Framework), set the **DataTypeCheckReportsRangeErrors** to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Formatting The Text

DES uses the System.Globalization.CultureInfo class to define most numeric formatting rules. You establish the desired CultureInfo on the **PeterBlum.DES.Globals.Page.CultureInfo** property. Currency uses the following properties from **CultureInfo.NumberFormat**:

- Decimal place character from CultureInfo.NumberFormat.CurrencyDecimalSeparator.
- Thousands separator from CultureInfo.NumberFormat.CurrencyGroupSeparator.
- Negative number format from CultureInfo.NumberFormat.CurrencyNegativePattern.
- The number of decimal places from CultureInfo.NumberFormat.CurrencyDecimalDigits.

Other Formatting Rules

If you want to show the currency symbol, set **ShowCurrencySymbol** to `true`.

If you want to omit decimal digits when the decimal digits are only "0" characters, set **HideDecimalWhenZero** to `true`.

To show thousands separators, set **ShowThousandsSeparator** to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Adding A Spinner

A Spinner is a control with two arrows that appear to the right of the textbox. For example:

When the user clicks on an arrow, it increments or decrements the value. If they hold down the button, it repeatedly changes the value and after 5 cycles, its speed increases.

To use the Spinner, set **ShowSpinner** to `true`. Use the **SpinnerManager** object to change the URL to the arrow button images and the auto repeat speed. (**SpinnerManager** is a property of **PeterBlum.DES.Globals.Page** and the PageManager control.)

Note: Spinners are only supported on these browsers: IE Windows 5+, Netscape 7+, Mozilla 1.1+, FireFox, Opera 7+, and Safari.

If you have client-side code that shows or hides the textbox, call the function `DES_Refresh()` to tell DES to change the visibility of the spinners. *This is handled automatically when using the FieldStateController.*

When the spinner is shown, the control attempts to keep the spinner side-by-side with the textbox by enclosing the two elements in either a `<span>` or `<table>` tag, depending on the browser. If you want to remove the enclosure, set **ContainerMode** to `None`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Connecting Data To Other Fields On The Client-Side

Often users want to update other textboxes and labels when a value is changed in a CurrencyTextBox. Use the CalculationController to perform math based on these textboxes. It can update a textbox or label for you. See the **Peter's Interactive Pages User's Guide**.

For other situations, write your own client-side JavaScript code. DES supplies a number of JavaScript functions to greatly simplify working with the CurrencyTextBoxes on the client-side. See "JavaScript Support Functions".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Adding a CurrencyTextBox

These steps ask you to jump around the document using clicks on links. Adobe Reader offers a **Previous View** command to return to the link. Look for this in the Adobe Reader (shown v6.0)

1. Prepare the page for DES controls. See "Preparing a page for DES controls" in the **General Features Guide**. It covers issues like style sheets, AJAX, and localization.
2. Add a CurrencyTextBox control to the page.

   Visual Studio and Visual Web Developer Design Mode Users

   Drag the CurrencyTextBox control from the Toolbox onto your web form.

   Text Entry Users

   Add the control (inside the <form> area):

   ```
   <des:CurrencyTextBox id="[YourControlID]" runat="server" />
   ```

   Programmatically creating the CurrencyTextBox control

   - Identify the control which you will add the CurrencyTextBox control to its **Controls** collection. Like all ASP.NET controls, the TextBox can be added to any control that supports child controls, like Panel, User Control, or TableCell. If you want to add it directly to the Page, first add a PlaceHolder at the desired location and use the PlaceHolder.
   - Create an instance of the CurrencyTextBox control class. The constructor takes no parameters.
   - Assign the **ID** property.
   - Add the CurrencyTextBox control to the **Controls** collection.

   In this example, the CurrencyTextBox is created with an **ID** of "CurrencyTextBox1". It is added to PlaceHolder1.

   [C#]

   ```
   PeterBlum.DES.CurrencyTextBox vTextBox = new PeterBlum.DES.CurrencyTextBox();
   vTextBox.ID = "CurrencyTextBox1";
   PlaceHolder1.Controls.Add(vTextBox);
   ```

   *Note: The namespace for these controls is PeterBlum.DES. If you prefer, add a **using** clause to that namespace on your form.*

   [VB]

   ```
   Dim vTextBox As PeterBlum.DES.CurrencyTextBox = _
      New PeterBlum.DES.CurrencyTextBox()
   vTextBox.ID = "CurrencyTextBox1"
   PlaceHolder1.Controls.Add(vTextBox)
   ```

   *Note: The namespace for these controls is PeterBlum.DES. If you prefer, add an **Imports** clause to that namespace on your form.*

   Guidelines for setting properties

   - Design mode users can use the Properties Editor or the Expanded Properties Editor. (See "Expanded Properties Editor" in the **General Features Guide**.) The SmartTag also offers some of the most important properties.
   - Text entry users should add the properties into the `<des:ControlClass>` tag in this format: `propertyname="value"`
   - When setting a property programmatically, have a reference to the control's object and set the property according to your language's rules.

3. Set the properties associated with the CurrencyTextBox. See "Properties for the CurrencyTextBox".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

4. Assign Validators to the TextBox.

Using DES Validation Framework

See the **Validation User's Guide** for details on these validators.

- Always add the DataTypeCheckValidator to block formatting errors.
- Add a RangeValidator when you are using the **MinValue** and **MaxValue** properties.
- If you have two CurrencyTextBoxes where one needs to be greater, less than, equal or not equal to the other, add a CompareTwoFieldsValidator.
- If you have two CurrencyTextBoxes where one must be greater or less than the other by a specific number, add a DifferenceValidator. Specify the number in the **DifferenceValue** property.
- Be sure that server side validation has been correctly set up.

Using Native Validation Framework

These validators are found in the **PeterBlum.DES.NativeValidators** assembly. Do not use the original CompareValidator and RangeValidator that come with ASP.NET because they don't handle the rich data entry formats of the CurrencyTextBox. See "Validation with the Native Validation Framework".

- Always add the CompareValidator with its **Operator** set to `DataTypeCheck` to block formatting errors.
- Add a RangeValidator when you are using the **MinValue** and **MaxValue** properties.
- If you have two CurrencyTextBoxes where one needs to be greater, less than, equal or not equal to the other, add a CompareValidator. Set its **ControlToValidate** property to the start CurrencyTextBox and **ControlToCompare** property to the end CurrencyTextBox.
- If you have two CurrencyTextBoxes where one must be greater or less than the other by a specific number, add the DifferenceValidator.
- Be sure that server side validation has been correctly set up.

5. Get and set the value using the **DoubleValue** or **DecimalValue** property. When databinding, use **DoubleBindable**.

   Usually you will set the initial value in the `Page_Load()` method when **Page.IsPostBack** is `false`. You will get the submitted value in your post back event method, after checking that **PeterBlum.DES.Globals.Page.IsValid** (DES validation framework) or **Page.IsValid** (native validation framework) is `true`.

6. Here are some other considerations:

   - If you are using an AJAX system to update this control, set the **InAJAXUpdate** property to `true`. Also make sure the PageManager control or AJAXManager object has been setup for AJAX. See "Using these Controls With AJAX" in the **General Features Guide**. *Failure to follow these directions can result in incorrect behavior and javascript errors.*
   - This control does not preserve most of its properties in the ViewState, to limit its impact on the page. If you need to use the ViewState to retain the value of a property, see "The ViewState and Preserving Properties for PostBack" in the **General Features Guide**.
   - If you encounter errors, see the "Troubleshooting" section for extensive topics based on several years of tech support's experience with customers.
   - See also "Additional Topics for Using These Controls".

Use demos here: http://www.peterblum.com/DES/DemoTextBoxes.aspx.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties for the CurrencyTextBox

Most of the properties are inherited from `PeterBlum.DES.TextBox` (see "Properties for the Enhanced TextBox") and the ASP.NET TextBox (see "System.Web.UI.WebControls.TextBox Members").

Click on any of these topics to jump to them:

- Getting And Setting The Value Properties
- Editing Properties
- Formatting Properties
- Spinner Properties

The following topics are inherited from DES's TextBox control:

- Editing Properties
- Behavior Properties
- Appearance Properties
- AutoPostBack Properties
- Value When Blank Properties
- ToolTip Properties
- Hint Properties
- Tab Rules Properties
- Client-Side Functions Properties

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Getting And Setting The Value Properties

There are numerous ways to get and set the numeric value of this textbox. Choose the best one for your needs.

Some of these properties are hidden from the Properties Editor as they are only intended to be used programmatically. When shown, they appear under the "Data" Category.

- **DoubleValue** (double) – Gets and sets the value for this control using a `System.Double` value. *See also DecimalValue and other properties in this section as alternatives.* You generally access this property programmatically. This property is preferred over using the **Text** property. However, you can get and set the value on the **Text** property so long as you handle the conversion between string and Double.

  Usually you will set the initial value in the `Page_Load()` method when **Page.IsPostBack** is `false`. You will get the submitted value in your post back event method, after checking that **PeterBlum.DES.Globals.Page.IsValid** (when using a DataTypeCheckValidator) or its own **IsValid** property is `true`.

  Note: If you attempt to get a value from DoubleValue when the text is incorrectly formatted, you will get a System.FormatException thrown.

  You can set the textbox value to "" either by assigning **Text** = "" or **DoubleValue** = `System.Double.MinValue`.

  Note: While .net handles larger values with its System.Decimal type, JavaScript on the client-side is limited to something close to the range of the System.Double type.

- **DecimalValue** (decimal) – Gets and sets the value for this control using a `System.Decimal` value. *See also DoubleValue and other properties in this section as alternatives.* You generally access this property programmatically. This property is preferred over using the **Text** property. However, you can get and set the value on the **Text** property so long as you handle the conversion between string and Decimal.

  Usually you will set the initial value in the `Page_Load()` method when **Page.IsPostBack** is `false`. You will get the submitted value in your post back event method, after checking that **PeterBlum.DES.Globals.Page.IsValid** (when using a DataTypeCheckValidator) or its own **IsValid** property is `true`.

  Note: If you attempt to get a value from DecimalValue when the text is incorrectly formatted, you will get a System.FormatException thrown.

  You can set the textbox value to "" either by assigning **Text** = "" or **DecimalValue** = `System.Decimal.MinValue` or call `SetEmpty()`.

  Note: While .net handles larger values with its System.Decimal type, on the client-side JavaScript is limited to something close to the range of the System.Double type.

- **DoubleBindable** (object) – An alternative to **DoubleValue** designed for handling multiple types. It accepts either a decimal, double, single, int32, SqlDecimal, SqlDouble, SqlInt32, `null`, string, or `System.DbNull.Value`.

  It returns a Double or SqlDouble depending on **BindableMode**.

  It is preferred when using DataBinding, due to its flexibility with data types.

  When evaluating a string, the format must represent a decimal or integer value respecting the culture formatting of **PeterBlum.DES.Globals.Page.CultureInfo**.

- **BindableMode** (enum PeterBlum.DES.BindMode) – Determines how the **DoubleBindable** property returns its value, as either a double or a `System.Data.SqlTypes.SqlDouble`.
  - `Normal` – Returns a double when the textbox is valid or `null`/`nothing` when the textbox is blank or has an illegal value. This is the default value.
  - `SqlTypes` – Returns a SqlDouble when the textbox is valid or `System.DBNull.Value` when the textbox is blank or has an illegal value.
  - `TwoWay` – Use this when you are using `System.Web.UI.WebControls.SqlDataSource` and other DataSource web controls to implement two-way databinding. Returns a double when the textbox is valid or `null`/`nothing` when the textbox is blank or has an illegal value.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **DoubleNullable** (double) – *Requires ASP.NET 2 or higher.* Read only. An alternative to **DoubleValue** that accepts either a double or `null`/`nothing`. When using `null`/`nothing`, it clears the textbox.
- **DoubleValueOrZero** (double) – An alternative to **DoubleValue** that returns a value even if the text is blank or not a valid number. When it's blank or an invalid entry, it returns a value of 0.0. This is a read-only property.
- **IsValid** (Boolean) – Determines if the contents of the TextBox represent a number. It returns `true` when it does represent a number. It always returns `false` when the **Text** property is blank (after trimming). Consider using a DataTypeCheckValidator instead of this property. This is a read only property.
- **IsEmpty** (Boolean) – Determines if the text is blank (after trimming). You should not attempt to get the field's value when this returns `true` because a blank textbox has no value. Consider adding a RequiredTextValidator to this control to prevent getting blank text fields. This is a read only property.
- **MinValue** (string) – Set the minimum value of a range. It affects the spinners, up/down arrow key commands, and is automatically used by the RangeValidator when its own **Minimum** property is unassigned.

  While it's a string type, it must represent an integer or double value. This property is great for design mode and ASP.NET Declarative Syntax. When programming, it's easier to use **MinValueAsDouble** because it doesn't require conversion from your double to a string.
- **MaxValue** (string) – Set the maximum value of a range. It affects the spinners, up/down arrow key commands, and is automatically used by the RangeValidator when its own **Maximum** property is unassigned.

  While it's a string type, it must represent an integer or double value. This property is great for design mode and ASP.NET Declarative Syntax. When programming, it's easier to use **MaxValueAsDouble** because it doesn't require conversion from your double to a string.
- **MinValueAsDouble** (double) – Set the minimum value of a range. Alternative to **MinValue** that takes a Double. Must be set programmatically. It affects the spinners, up/down arrow key commands, and is automatically used by the RangeValidator when its own **Minimum** property is unassigned.
- **MaxValueAsDouble** (double) – Set the maximum value of a range. Alternative to **MaxValue** that takes a Double. Must be set programmatically. It affects the spinners, up/down arrow key commands, and is automatically used by the RangeValidator when its own **Maximum** property is unassigned.
- **AllowNegatives** (Boolean) – Determines if negative numbers are permitted. When `true`, they are permitted. It defaults to `true`. When `false`, keyboard filtering will not allow the minus ("-") character, and the DataTypeCheckValidator will report errors when negative values are entered. It defaults to `false`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Editing Properties

The Properties Editor shows these properties in the category "Editing".

- **ReadOnly** (Boolean) – Determines if the textbox is editable or not. When **ReadOnly** is `true`, it is not editable. However, the focus can enter the textbox. You can still permit editing by using the arrow keys (**UpDnKeysIncrement** = `true`) and with the spinners (**ShowSpinners** = `true`) by setting **ReadOnlyAllowsEdits** to `true`.
- **ReadOnlyAllowsEdits** (Boolean) – Overrides the **ReadOnly** property to let spinners and arrow keys edit a read-only textbox. It defaults to `false`.

  If you don't want to support arrows, set **UpDnKeysIncrement** to `false`.
- **UpDnKeysIncrement** (Boolean) – Determines if the up and down arrow keys increment and decrement the value.

  When `true`, the up and down arrow keys increment and decrement. When `false`, they do not.

  It defaults to `true`.
- **AcceptPeriodAsDecimalSeparator** (Boolean) – Many cultures do not use the period as the decimal separator. It makes numeric entry from the numeric keypad more difficult, because it features a period key. When setup, both the culture's decimal separator and a period are allowed as the decimal separator. The textbox will convert the period to the decimal character.

  Does not apply for cultures that already use a period for the decimal separator. Some cultures use a period as a thousands separator. In those cases, the parser will only consider periods the decimal separator when there is only one period character and the culture's decimal separator is not found.

  When `true`, the feature is used so long as the current culture's decimal separator is not already a period.

  When `false`, the feature is not used.

  It defaults to `false`.
- **UseKeyboardFiltering** (Boolean) – When `true`, the browser filters out characters that are not supported by the data type. When `false`, it allows all keystrokes. It defaults to `true`. Keyboard filtering is automatically disabled on browsers that do not support DES's client-side code.

  The filter for CurrencyTextBox allows only digits, the Currency Symbol (**CultureInfo.NumberFormat.CurrencySymbol**), the culture specific decimal point character (**CultureInfo.NumberFormat.NumberDecimalSeparator**), minus ("-") and parenthesis. Minus and parenthesis depend on the **AllowNegatives** property. If **UseCurrencySymbol** is `true`, it also permits the currency symbol in CultureInfo.NumberFormat.CurrencySymbol.
- **OnChangeFunctionName** (string) – Allows you to extend the client side **onchange** event fired when the user exits the CurrencyTextBox or invokes one of its commands. During the **onchange** event, the number is validated, reformatted, and passed along to other objects.

  You create a JavaScript function and assign its name to the **OnChangeFunctionName** property. You can give it any legal JavaScript name you want.

  ALERT: Many users make the mistake of assigning JavaScript code to this property. This will cause JavaScript errors. GOOD: "MyFunction". BAD: "MyFunction();" and "alert('stop it')".

  Note: JavaScript is case sensitive. Be sure the value of this property exactly matches the function definition.

  See also "Adding Your JavaScript to the Page".

Defining the JavaScript Function

Declare a JavaScript function in your page with three parameters:

- TextBoxID (string) - The element ID to the text box of the date text box control. It matches the **ClientID** property of the TextBox.
- Value (floating point) - The resulting number in the field. It will be `null` if the number was invalid or the textbox was blank.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- Error (boolean) – When `true`, the number was invalid including a bad format or an illegal number. When `false`, there is no error or the textbox is blank.

The function does not return anything.

Here is an example function which assigns the number to another CurrencyTextBox whose ID is CTB2:

```
function MyOnChangeFnc(pTBId, pValue, pError)
{
   if (!pError)
      DES_SetDTTBValue('CTB2', pValue, true);  // if pValue=null, CTB2 is blank
}
```

- **OnChangeFunctionAlways** (boolean) – Used when **OnChangeFunctionName** is defined. When **OnChangeFunctionAlways** is `false`, your function will only be called when there is a valid number. When **OnChangeFunctionAlways** is `true`, it will be called on every change. You can detect these cases:
  - Valid number: *Value* parameter is an integer and *Error* is `false`.
  - Invalid number: *Value* parameter is `null` and *Error* is `true`.
  - Blank textbox: *Value* parameter is `null` and *Error* is `false`.

See "JavaScript Support Functions" for numerous functions that can assist in your coding efforts.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Formatting Properties

These properties determine what text is considered valid for a decimal number. In addition, the **PeterBlum.DES.Globals.Page.CultureInfo** determines many formatting rules.

- **ShowThousandsSeparator** (Boolean) – When `true`, the thousand separator is added into the textbox as it is reformatted. The thousands separator character is defined in **PeterBlum.DES.Globals.Page.CultureInfo**. When `false`, no thousand characters are shown. If the user had entered them, they would be removed. It defaults to `false`.

  Note: This TextBox always permits entry of the thousands separator regardless of this property.

- **UseCurrencySymbol** (Boolean) – When `true`, the user can enter the currency symbol and when the field is reformatted, the symbol is automatically added. When `false`, no currency symbol is permitted. It defaults to `false`.

  The currency symbol is defined in the CultureInfo.NumberFormat. Reformatting will always place it in the correct position for the culture (before or after the number; inside or outside the negative symbol.)

- **AllowExtraDecimalDigits** (Boolean) – When `true`, the user can enter more decimal digits than defined by the **CultureInfo.NumberFormat.CurrencyDecimalDigits**. When `false`, an error is reported when the decimal digits exceeds **CultureInfo.NumberFormat.CurrencyDecimalDigits**. It defaults to `false`.

- **HideDecimalWhenZero** (Boolean) – When `true`, a number with only zeros in the decimal portion omits the decimal portion when the value is reformatted. Useful when large currencies are expected. For example, text entered as “12.0” is reformatted “12” while “12.1” is reformatted to “12.10”. It defaults to `false`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Spinner Properties

The Spinner is an extension to the CurrencyTextBox. It provides a pair of arrow buttons that increment or decrement the value of the textbox when clicked. You can customize the button appearance and autorepeat speed with properties on **PeterBlum.DES.Globals.Page.SpinnerManager**. It respects the limits established with the **MinValue** and **MaxValue** properties.

The following properties are identified in the category "Spinner" under the Properties Editor:

- **ShowSpinner** (Boolean) – When `true`, the spinner control is shown. When `false`, it is not. It defaults to `false`.
- **IncrementValue** (Double) – The number to add or subtract to the current value. It supports decimal values. It defaults to 1.0.
- **ContainerMode** (enum PeterBlum.DES.ContainerMode) – When the spinner is shown, the HTML output looks like this:

  ```
  <input type='text' [attributes]/><table>[spinner buttons]<table/>
  ```

  When you assign absolute positioning ("gridlayout"), the style attribute used for absolute positioning would normally get assigned to the `<input>` tag, only affecting its position. If nothing else was done, you would see the textbox in the correct location while the buttons would be elsewhere on the page.

  In addition, these tags can wrap around, especially when placed in a `<p>` tag or a table cell that is too small. The spinners appear below the textbox when that happens.

  Use the **ContainerMode** property to add either a `<span>` or `<table>` tag around this control that limits wrap around. This property is only used when you are showing spinners. When using absolute positioning, the styles used for positioning are moved into the container tag so all of this control's tags are grouped under a common position.

  The enumerated type `PeterBlum.DES.ContainerMode` has these values:

  - `None` – Do not create a container. Use it when you want to create your own container or otherwise control the formatting. If you need to absolutely position this control, you must assign the styles for absolute positioning to your own container.
  - `Auto` – Creates either a `<span>` or `<table>` depending on the browser's support of these styles: `white-space:nowrap` (used by `<span>`) and `display:inline-block` (used by `<table>`). Internet Explorer 5+ and the Mozilla-based browsers support `white-space:nowrap` (as determined by the **PeterBlum.DES.Globals.Page.Browser.MakeWhiteSpaceNoWrap** property.) Opera 7+ and Safari support `display:inline-block` (as determined by the **PeterBlum.DES.Globals.Page.Browser.MakeBlockInlineValue** property.)

    This is the default.

    Note: A tag generates much smaller HTML than the <table> tag.
  - `Span` – Creates a `<span>` tag. It uses the style `white-space:nowrap` to prevent wrapping. Some browsers do not support this style and it will be omitted, leaving the control susceptible to wrapping on those browsers. (The **PeterBlum.DES.Globals.Page.Browser.MakeWhiteSpaceNoWrap** property determines support for this style.)

    The `<span>` tag will also be assigned the style `vertical-align:text-bottom` to keep all elements in a reasonable layout. Yet, some browsers do not make the alignments for this style as well as Internet Explorer and Mozilla-based browsers.
  - `Table` – Creates a `<table>` around the controls with each HTML element in its own table cell. While this is the most reliable, it has these limitations on browsers other than Internet Explorer for Windows, Mozilla, FireFox, and Netscape 6+:

    Some browsers do not support the `display:inline` or `display:inline-block` style required to position the table inside a row of text. A `<span>` tag will be generated instead.

    It uses the style `vertical-align:middle` to attempt to align all elements. Yet, some browsers do not make the alignments for this style as well as Internet Explorer and Mozilla-based browsers.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

It sets `padding` and `margin` styles to 0px on all sides of the cells. It sets `cellpadding` and `cellspacing` to 0 as well. These attempt to remove any shifting imposed by the table's margins. Yet, some browsers do not make the positioning as well as Internet Explorer and Mozilla-based browsers. You may notice misalignment with other elements in the same row.

Note: A tag generates much smaller HTML than the <table> tag.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

PercentTextBox Control

The `PeterBlum.DES.PercentTextBox` is a TextBox designed for percent data entry. It knows how to convert a `System.Double` and `System.Integer` types into text and back. It has properties to determine the number of decimal places or even restrict to integer entry, if it supports negative numbers and shows the percent symbol. You can add a spinner control for the user to increment the value.

Click on any of these topics to jump to them:

- Features
- Using the PercentTextBox
  - Getting and Setting the Value of the TextBox
  - Formatting The Text
  - Adding A Spinner
  - Connecting Data To Other Fields On The Client-Side
  - When the TextBox is Empty
  - Validation on AutoPostBack
  - Interactive Hints
  - AutoComplete and “Smart Change System”
  - Other Behaviors
- Adding a PercentTextBox
- Properties for the PercentTextBox

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Features

Use demos here: http://www.peterblum.com/DES/DemoTextBoxes.aspx.

- It subclasses from `PeterBlum.DES.TextBox`, inheriting all of its qualities. See "Enhanced TextBox".
- Handles integer and decimal formats. You can lock out decimal entry if you like.
- It optionally includes the percent symbol in the text.
- Get and set the value of the textbox using an integer, double, or decimal data type instead of a string, avoiding you having to write conversion code.
- Set a property to allow or prevent entry of negative numbers.
- It is culture sensitive, respecting the CultureInfo object defined on the **PeterBlum.DES.Globals.Page.CultureInfo** property.
- On the client-side, it filters keystrokes so users can only enter characters suitable to for integer entry.
- On the client-side, it reformats if needed as the user exits the textbox. For example, if the user enters "1000", it reformats to "1,000" (when showing the optional thousands separator.)
- When you attach a Validator to them, the Validator automatically configures itself to evaluate an integer with the other data entry rules specified on this control.
- Use a spinner control (up and down arrows to the right of the textbox) to increment the value.
- Use up and down arrow keys to increment the value.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the PercentTextBox

Use demos here: http://www.peterblum.com/DES/DemoTextBoxes.aspx.

The `PeterBlum.DES.PercentTextBox` is subclassed from `PeterBlum.DES.TextBox`, which is an enhanced version of the TextBox control supplied with ASP.NET. If you know how to use the ASP.NET TextBox, you already know how to use this control. See "Using the Enhanced TextBox Control".

Click on any of these topics to jump to them:

- Getting and Setting the Value of the TextBox
- Formatting The Text
- Adding A Spinner
- Connecting Data To Other Fields On The Client-Side

The following topics are inherited from DES's TextBox control:

- When the TextBox is Empty
- Validation on AutoPostBack
- Interactive Hints
- AutoComplete and "Smart Change System"
- Other Behaviors

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Getting and Setting the Value of the TextBox

When getting or setting the value, use the **IntegerValue, DoubleValue**, or **DecimalValue** property instead of the **Text** property. These properties convert your integer or double into the localized text shown in the textbox.

If you are using DataBinding, consider these alternative properties: **DoubleBindable** and **IntegerBindable**.

By default, whole number entry is offered. If you want decimal entry set **WholeNumbersOnly** to `false` and limit the number of decimal places with **MaxDecimalPlaces**.

By default, negatives are not permitted. If allow negative percentages, set **AllowNegatives** to `true`. The DataTypeCheckValidator will respect this setting and report an error when a negative number is found.

You can check for a blank textbox by checking the **IsEmpty** property for `true`. The PercentTextBox also contain an invalid entry, such as an illegal characters or badly formatted entry. You can check for an invalid entry by checking the **IsValid** property for `false`, although a validator should avoid having to test for an invalid value.

When you need a server side event that is fired when the textbox's value has changed, use the **TextChanged** event.

Data Entry Validation

Consider validation a manditory part of data entry, whether it's a PercentTextBox or just an ordinary textbox. It prevents illegal entries from getting into your database. Even if the PercentTextBox enforces some rules for you, users may not have a browser that supports the client-side code for the PercentTextBox, or its validators for that matter. Hackers often turn off javascript in their browser in hopes that your server side code doesn't protect against their illegal data.

Validation Guidelines:

- At minimum, validate the percentage number format with one of these validators.
    - When using the DES Validation Framework, assign a DataTypeCheckValidator to each PercentTextBox. (See the **Validation User's Guide**.)
    - When using the Native Validation Framework, use DES's CompareValidator with the **Operator** property set to `DataTypeCheck`. (This validator is in the **PeterBlum.DES.NativeValidators** assembly. If you haven't done so, add this assembly to its own tab in the Visual Studio/VWD toolbox. See the **Installation Guide** for details.) See "Validation with the Native Validation Framework".
- Set up server side validation.
    - When using DES's validation framework, test **PeterBlum.DES.Globals.Page.IsValid** in your postback event handler methods. Only use the data if it is `true`.
    - When using the native validation framework, test **Page.IsValid** in your postback event handler. Only use the data if it is `true`.
- You can establish a numeric range to prevent selection outside the range using **MinValue** and **MaxValue**. By default, these establish the range of 0 to 100. Add the RangeValidator to report errors when the user types numbers outside the range. DES provides two RangeValidators, one for its own validation framework and the other for the native validation framework. Choose the correct one. *Do not use the original RangeValidator that comes with ASP.NET.*

  Alternatively, if you feel an out of range value can be reported under the error message from your DataTypeCheckValidator (or CompareValidator in the Native Validation Framework), set the **DataTypeCheckReportsRangeErrors** to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Formatting The Text

DES uses the System.Globalization.CultureInfo class to define most numeric formatting rules. You establish the desired CultureInfo on the **PeterBlum.DES.Globals.Page.CultureInfo** property. Percentages use the following values from **CultureInfo.NumberFormat**:

- Decimal place character from CultureInfo.NumberFormat.NumberDecimalSeparator. *Not PercentDecimalSeparator.*
- Thousands separator from CultureInfo.NumberFormat.NumberGroupSeparator. *Not PercentGroupSeparator.*
- Positive and negative number format when showing a percent symbol from CultureInfo.NumberFormat.PercentPositivePattern and CultureInfo.NumberFormat.PercentNegativePattern. When not showing a percent symbol, from CultureInfo.NumberFormat.NumberPositivePattern and CultureInfo.NumberFormat.NumberNegativePattern.

Other Formatting Rules

To show the percent symbol, set **UsePercentSymbol** to `true`.

To show thousands separators, set **ShowThousandsSeparator** to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Adding A Spinner

A Spinner is a control with two arrows that appear to the right of the textbox. For example:

When the user clicks on an arrow, it increments or decrements the value. If they hold down the button, it repeatedly changes the value and after 5 cycles, its speed increases.

To use the Spinner, set **ShowSpinner** to `true`. Use the **SpinnerManager** object to change the URL to the arrow button images and the auto repeat speed. (**SpinnerManager** is a property of **PeterBlum.DES.Globals.Page** and the PageManager control.)

Note: Spinners are only supported on these browsers: IE Windows 5+, Netscape 7+, Mozilla 1.1+, FireFox, Opera 7+, and Safari.

If you have client-side code that shows or hides the textbox, call the function `DES_Refresh()` to tell DES to change the visibility of the spinners. *This is handled automatically when using the FieldStateController.*

When the spinner is shown, the control attempts to keep the spinner side-by-side with the textbox by enclosing the two elements in either a `<span>` or `<table>` tag, depending on the browser. If you want to remove the enclosure, set **ContainerMode** to `None`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Connecting Data To Other Fields On The Client-Side

Often users want to update other textboxes and labels when a value is changed in a PercentTextBox. Use the CalculationController to perform math based on these textboxes. It can update a textbox or label for you. See the **Peter's Interactive Pages User's Guide**.

For other situations, write your own client-side JavaScript code. DES supplies a number of JavaScript functions to greatly simplify working with the PercentTextBoxes on the client-side. See "JavaScript Support Functions".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Adding a PercentTextBox

These steps ask you to jump around the document using clicks on links. Adobe Reader offers a **Previous View** command to return to the link. Look for this in the Adobe Reader (shown v6.0)

1. Prepare the page for DES controls. See "Preparing a page for DES controls" in the **General Features Guide**. It covers issues like style sheets, AJAX, and localization.
2. Add a PercentTextBox control to the page.

Visual Studio and Visual Web Developer Design Mode Users

Drag the PercentTextBox control from the Toolbox onto your web form.

Text Entry Users

- Add the control (inside the <form> area):

```
<des:PercentTextBox id="[YourControlID]" runat="server" />
```

Programmatically creating the PercentTextBox control

- Identify the control which you will add the PercentTextBox control to its **Controls** collection. Like all ASP.NET controls, the TextBox can be added to any control that supports child controls, like Panel, User Control, or TableCell. If you want to add it directly to the Page, first add a PlaceHolder at the desired location and use the PlaceHolder.
- Create an instance of the PercentTextBox control class. The constructor takes no parameters.
- Assign the **ID** property.
- Add the PercentTextBox control to the **Controls** collection.

In this example, the PercentTextBox is created with an **ID** of "PercentTextBox1". It is added to PlaceHolder1.

[C#]

```
PeterBlum.DES.PercentTextBox vTextBox = new PeterBlum.DES.PercentTextBox();
vTextBox.ID = "PercentTextBox1";
PlaceHolder1.Controls.Add(vTextBox);
```

*Note: The namespace for these controls is PeterBlum.DES. If you prefer, add a **using** clause to that namespace on your form.*

[VB]

```
Dim vTextBox As PeterBlum.DES.PercentTextBox =  _
   New PeterBlum.DES.PercentTextBox()
vTextBox.ID = "PercentTextBox1"
PlaceHolder1.Controls.Add(vTextBox)
```

*Note: The namespace for these controls is PeterBlum.DES. If you prefer, add an **Imports** clause to that namespace on your form.*

Guidelines for setting properties

- Design mode users can use the Properties Editor or the Expanded Properties Editor. (See "Expanded Properties Editor" in the **General Features Guide**.) The SmartTag also offers some of the most important properties.
- Text entry users should add the properties into the `<des:ControlClass>` tag in this format: `propertyname="value"`
- When setting a property programmatically, have a reference to the control's object and set the property according to your language's rules.

3. Set the properties associated with the PercentTextBox. See "Properties for the PercentTextBox".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

4. By default, a range is established of 0 to 100. If you need to change the range, use **MinValue** and **MaxValue** properties to the range.
5. Assign Validators to the TextBox.

Using DES Validation Framework

See the **Validation User's Guide** for details on these validators.

- Always add the DataTypeCheckValidator to block formatting errors.
- Add a RangeValidator unless you have disabled the **MinValue** and **MaxValue** properties.
- If you have two PercentTextBoxes where one needs to be greater, less than, equal or not equal to the other, add a CompareTwoFieldsValidator.
- If you have two PercentTextBoxes where one must be greater or less than the other by a specific number, add a DifferenceValidator. Specify the number in the **DifferenceValue** property.
- Be sure that server side validation has been correctly set up.

Using Native Validation Framework

These validators are found in the **PeterBlum.DES.NativeValidators** assembly. Do not use the original CompareValidator and RangeValidator that come with ASP.NET because they don't handle the rich data entry formats of the PercentTextBox. See "Validation with the Native Validation Framework".

- Always add the CompareValidator with its **Operator** set to `DataTypeCheck` to block formatting errors.
- Add a RangeValidator unless you have disabled the **MinValue** and **MaxValue** properties.
- If you have two PercentTextBoxes where one needs to be greater, less than, equal or not equal to the other, add a CompareValidator. Set its **ControlToValidate** property to the start PercentTextBox and **ControlToCompare** property to the end PercentTextBox.
- If you have two PercentTextBoxes where one must be greater or less than the other by a specific number, add the DifferenceValidator.
- Be sure that server side validation has been correctly set up.

6. Get and set the value using the **IntegerValue**, **DoubleValue** or **DecimalValue** property. When databinding, use **IntegerBindable** or **DoubleBindable**.

   Usually you will set the initial value in the `Page_Load()` method when **Page.IsPostBack** is `false`. You will get the submitted value in your post back event method, after checking that **PeterBlum.DES.Globals.Page.IsValid** (DES validation framework) or **Page.IsValid** (native validation framework) is `true`.

7. Here are some other considerations:
   - If you are using an AJAX system to update this control, set the **InAJAXUpdate** property to `true`. Also make sure the PageManager control or AJAXManager object has been setup for AJAX. See "Using these Controls With AJAX" in the **General Features Guide**. *Failure to follow these directions can result in incorrect behavior and javascript errors.*
   - This control does not preserve most of its properties in the ViewState, to limit its impact on the page. If you need to use the ViewState to retain the value of a property, see "The ViewState and Preserving Properties for PostBack" in the **General Features Guide**.
   - If you encounter errors, see the "Troubleshooting" section for extensive topics based on several years of tech support's experience with customers.
   - See also "Additional Topics for Using These Controls".

Use demos here: http://www.peterblum.com/DES/DemoTextBoxes.aspx.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties for the PercentTextBox

Most of the properties are inherited from `PeterBlum.DES.TextBox` (see "Properties for the Enhanced TextBox") and the ASP.NET TextBox (see "System.Web.UI.WebControls.TextBox Members").

Click on any of these topics to jump to them:

- Getting And Setting The Value Properties
- Editing Properties
- Formatting Properties
- Spinner Properties

The following topics are inherited from DES's TextBox control:

- Editing Properties
- Behavior Properties
- Appearance Properties
- AutoPostBack Properties
- Value When Blank Properties
- ToolTip Properties
- Hint Properties
- Tab Rules Properties
- Client-Side Functions Properties

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Getting And Setting The Value Properties

There are numerous ways to get and set the numeric value of this textbox. Choose the best one for your needs.

Some of these properties are hidden from the Properties Editor as they are only intended to be used programmatically. When shown, they appear under the "Data" Category.

- **IntegerValue** (integer) – Gets and sets the value for this control using a `System.Int32` value. *See also DoubleValue, DecimalValue, and other properties that follow as alternatives.* You generally access this property programmatically. This property is preferred over using the **Text** property. However, you can get and set the value on the **Text** property so long as you handle the conversion between string and integer.

  Usually you will set the initial value in the `Page_Load()` method when **Page.IsPostBack** is `false`. You will get the submitted value in your post back event method, after checking that **PeterBlum.DES.Globals.Page.IsValid** (when using a DataTypeCheckValidator) or its own **IsValid** property is `true`.

  Note: If you attempt to get a value from IntegerValue when the text is incorrectly formatted, you will get a System.FormatException thrown.

  You can set the textbox value to "" either by assigning **Text** = "" or **IntegerValue** = `System.Int32.MinValue` or call `SetEmpty()`.

- **DoubleValue** (double) – Gets and sets the value for this control using a `System.Double` value. *See also DecimalValue, IntegerValue, and other properties in this section as alternatives.* You generally access this property programmatically. This property is preferred over using the **Text** property. However, you can get and set the value on the **Text** property so long as you handle the conversion between string and Double.

  Usually you will set the initial value in the `Page_Load()` method when **Page.IsPostBack** is `false`. You will get the submitted value in your post back event method, after checking that **PeterBlum.DES.Globals.Page.IsValid** (when using a DataTypeCheckValidator) or its own **IsValid** property is `true`.

  Note: If you attempt to get a value from DoubleValue when the text is incorrectly formatted, you will get a System.FormatException thrown.

  You can set the textbox value to "" either by assigning **Text** = "" or **DoubleValue** = `System.Double.MinValue` or call `SetEmpty()`.

  Note: While .net handles larger values with its System.Decimal type, on the client-side JavaScript is limited to something close to the range of the System.Double type.

- **DecimalValue** (decimal) – Gets and sets the value for this control using a `System.Decimal` value. *See also DoubleValue, IntegerValue, and other properties in this section as alternatives.* You generally access this property programmatically. This property is preferred over using the **Text** property. However, you can get and set the value on the **Text** property so long as you handle the conversion between string and Decimal.

  Usually you will set the initial value in the `Page_Load()` method when **Page.IsPostBack** is `false`. You will get the submitted value in your post back event method, after checking that **PeterBlum.DES.Globals.Page.IsValid** (when using a DataTypeCheckValidator) or its own **IsValid** property is `true`.

  Note: If you attempt to get a value from DecimalValue when the text is incorrectly formatted, you will get a System.FormatException thrown.

  You can set the textbox value to "" either by assigning **Text** = "" or **DecimalValue** = `System.Decimal.MinValue` or call `SetEmpty()`.

  Note: While .net handles larger values with its System.Decimal type, on the client-side JavaScript is limited to something close to the range of the System.Double type.

- **IntegerValueOrZero** (double) – An alternative to **IntegerValue** that returns a value even if the text is blank or not a valid number. When its blank or an invalid entry, it returns a value of 0. This is a read-only property.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **IntegerBindable** (object) – An alternative to **IntegerValue** designed for handling multiple types. It accepts either a byte, sbyte, int16, int32, SqlInt16, SqlInt32, `null`, string, or `System.DbNull.Value`.

  It returns a integer (System.Int32) or Sql32 depending on **BindableMode**.

  It is preferred when using DataBinding, due to its flexibility with data types.

  When evaluating a string, the format must represent an integer value respecting the culture formatting of **PeterBlum.DES.Globals.Page.CultureInfo**.

- **DoubleBindable** (object) – An alternative to **DoubleValue** designed for handling multiple types. It accepts either a decimal, double, single, int32, SqlDecimal, SqlDouble, SqlInt32, `null`, string, or `System.DbNull.Value`.

  It returns a Double or SqlDouble depending on **BindableMode**.

  It is preferred when using DataBinding, due to its flexibility with data types.

  When evaluating a string, the format must represent a decimal or integer value respecting the culture formatting of **PeterBlum.DES.Globals.Page.CultureInfo**.

- **BindableMode** (enum PeterBlum.DES.BindMode) – Determines how the **IntegerBindable** and **DoubleBindable** properties returns its value, as either a native type (32 bit integer or double) or a `System.Data.SqlTypes` type (SqlInteger or SqlDouble).
  - `Normal` – Returns an integer or double when the textbox is valid or `null`/`nothing` when the textbox is blank or has an illegal value. This is the default value.
  - `SqlTypes` – Returns a SqlInteger or SqlDouble when the textbox is valid or `System.DBNull.Value` when the textbox is blank or has an illegal value.
  - `TwoWay` – Use this when you are using `System.Web.UI.WebControls.SqlDataSource` and other DataSource web controls to implement two-way databinding. Returns an integer or double when the textbox is valid or `null`/`nothing` when the textbox is blank or has an illegal value.

- **IntegerNullable** (integer) – *Requires ASP.NET 2 or higher.* Read only. An alternative to **IntegerValue** that excepts either an integer (System.Int32) or `null`/`nothing`. When using `null`/`nothing`, it clears the textbox.

- **DoubleNullable** (double) – *Requires ASP.NET 2 or higher.* Read only. An alternative to **DoubleValue** that excepts either a double or `null`/`nothing`. When using `null`/`nothing`, it clears the textbox.

- **IsValid** (Boolean) – Determines if the contents of the TextBox represent a number. It returns `true` when it does represent a number. It always returns `false` when the **Text** property is blank (after trimming). Consider using a DataTypeCheckValidator instead of this property. This is a read only property.

- **IsEmpty** (Boolean) – Determines if the text is blank (after trimming). You should not attempt to get the field's value when this returns `true` because a blank textbox has no value. Consider adding a RequiredTextValidator to this control to prevent getting blank text fields. This is a read only property.

- **WholeNumbersOnly** (boolean) – Determines if the PercentTextbox only accepts and displays the whole number, with Percent formatting.

  If you assign a value that has decimal places, they are stripped, leaving only the whole number part. No rounding is applied. If you need a specific type of rounding, do it before assigning the value to this control.

  When `true`, decimals are prevented, both in keyboard entry and formatting.

  When `false`, decimals are allowed.

  It defaults to `true`.

- **MinValue** (string) – Set the minimum value of a range. It affects the spinners, up/down arrow key commands, and is automatically used by the RangeValidator when its own **Minimum** property is unassigned.

  While it's a string type, it must represent an integer or double value. This property is great for design mode and ASP.NET Declarative Syntax. When programming, it's easier to use **MinValueAsInteger** or **MinValueAsDouble** because they don't require conversion from your integer or double to a string.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **MaxValue** (string) – Set the maximum value of a range. It affects the spinners, up/down arrow key commands, and is automatically used by the RangeValidator when its own **Maximum** property is unassigned.

  While it's a string type, it must represent an integer or double value. This property is great for design mode and ASP.NET Declarative Syntax. When programming, it's easier to use **MaxValueAsInteger** or **MaxValueAsDouble** because they don't require conversion from your integer or double to a string.

- **MinValueAsInteger** (Integer) – Set the minimum value of a range. Alternative to **MinValue** that takes an integer. Must be set programmatically. It affects the spinners, up/down arrow key commands, and is automatically used by the RangeValidator when its own **Minimum** property is unassigned.

- **MaxValueAsInteger** (Integer) – Set the maximum value of a range. Alternative to **MaxValue** that takes an integer. Must be set programmatically. It affects the spinners, up/down arrow key commands, and is automatically used by the RangeValidator when its own **Maximum** property is unassigned.

- **MinValueAsDouble** (double) – Set the minimum value of a range. Alternative to **MinValue** that takes a Double. Must be set programmatically. It affects the spinners, up/down arrow key commands, and is automatically used by the RangeValidator when its own **Minimum** property is unassigned.

- **MaxValueAsDouble** (double) – Set the maximum value of a range. Alternative to **MaxValue** that takes a Double. Must be set programmatically. It affects the spinners, up/down arrow key commands, and is automatically used by the RangeValidator when its own **Maximum** property is unassigned.

- **AllowNegatives** (Boolean) – Determines if negative numbers are permitted. When `true`, they are permitted. It defaults to `true`. When `false`, keyboard filtering will not allow the minus ("-") character, and the DataTypeCheckValidator will report errors when negative values are entered. It defaults to `false`.

- **MaxDecimalPlaces** (Integer) – Determines the maximum number of decimal places allowed. When the user enters more, it is an error which the DataTypeCheckValidator will detect.

  Set to 0 to ignore this property. It defaults to 0.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Editing Properties

The Properties Editor shows these properties in the category "Editing".

- **ReadOnly** (Boolean) – Determines if the textbox is editable or not. When **ReadOnly** is `true`, it is not editable. However, the focus can enter the textbox. You can still permit editing by using the arrow keys (**UpDnKeysIncrement** = `true`) and with the spinners (**ShowSpinners** = `true`) by setting **ReadOnlyAllowsEdits** to `true`.
- **ReadOnlyAllowsEdits** (Boolean) – Overrides the **ReadOnly** property to let spinners and arrow keys edit a read-only textbox. It defaults to `false`.

  If you don't want to support arrows, set **UpDnKeysIncrement** to `false`.
- **UpDnKeysIncrement** (Boolean) – Determines if the up and down arrow keys increment and decrement the value.

  When `true`, the up and down arrow keys increment and decrement. When `false`, they do not.

  It defaults to `true`.
- **AcceptPeriodAsDecimalSeparator** (Boolean) – Many cultures do not use the period as the decimal separator. It makes numeric entry from the numeric keypad more difficult, because it features a period key. When setup, both the culture's decimal separator and a period are allowed as the decimal separator. The textbox will convert the period to the decimal character.

  Does not apply for cultures that already use a period for the decimal separator. Some cultures use a period as a thousands separator. In those cases, the parser will only consider periods the decimal separator when there is only one period character and the culture's decimal separator is not found.

  When `true`, the feature is used so long as the current culture's decimal separator is not already a period.

  When `false`, the feature is not used.

  It defaults to `false`.
- **UseKeyboardFiltering** (Boolean) – When `true`, the browser filters out characters that are not supported by the data type. When `false`, it allows all keystrokes. It defaults to `true`. Keyboard filtering is automatically disabled on browsers that do not support DES's client-side code.

  The filter for PercentTextBox allows only digits, the Percent Symbol (**CultureInfo.NumberFormat.PercentSymbol**), the culture specific decimal point character (**CultureInfo.NumberFormat.NumberDecimalSeparator**) and minus ("-"). Minus depends on the **AllowNegatives** property. Percent symbol depends on the **UsePercentSymbol** property.
- **OnChangeFunctionName** (string) – Allows you to extend the client side **onchange** event fired when the user exits the PercentTextBox or invokes one of its commands. During the **onchange** event, the number is validated, reformatted, and passed along to other objects.

  You create a JavaScript function and assign its name to the **OnChangeFunctionName** property. You can give it any legal JavaScript name you want.

  ALERT: Many users make the mistake of assigning JavaScript code to this property. This will cause JavaScript errors. GOOD: "MyFunction". BAD: "MyFunction();" and "alert('stop it')".

  Note: JavaScript is case sensitive. Be sure the value of this property exactly matches the function definition.

  See also "Adding Your JavaScript to the Page".

Defining the JavaScript Function

Declare a JavaScript function in your page with three parameters:

- TextBoxID (string) - The element ID to the text box of the date text box control. It matches the **ClientID** property of the TextBox.
- Value (floating point) - The resulting number in the field. It will be `null` if the number was invalid or the textbox was blank.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- Error (boolean) – When `true`, the number was invalid including a bad format or an illegal number. When `false`, there is no error or the textbox is blank.

The function does not return anything.

Here is an example function which assigns the number to another PercentTextBox whose ID is PTB2:

```
function MyOnChangeFnc(pTBId, pValue, pError)
{
   if (!pError)
      DES_SetDTTBValue('PTB2', pValue, true);  // if pValue=null, PTB2 is blank
}
```

- **OnChangeFunctionAlways** (boolean) – Used when **OnChangeFunctionName** is defined. When **OnChangeFunctionAlways** is `false`, your function will only be called when there is a valid number. When **OnChangeFunctionAlways** is `true`, it will be called on every change. You can detect these cases:
  - Valid number: *Value* parameter is an integer and *Error* is `false`.
  - Invalid number: *Value* parameter is `null` and *Error* is `true`.
  - Blank textbox: *Value* parameter is `null` and *Error* is `false`.

See "JavaScript Support Functions" for numerous functions that can assist in your coding efforts.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Formatting Properties

These properties determine what text is considered valid for a decimal number. In addition, the **PeterBlum.DES.Globals.Page.CultureInfo** determines many formatting rules.

- **UsePercentSymbol** (Boolean) – Determines if the percent symbol can appear in the text.

  When `true`, the percent symbol appears upon reformat and is allowed during entry. **PeterBlum.DES.Globals.Page.CultureInfo.NumberFormat.PercentSymbol** defines the character representing the percent symbol. Reformatting follows the **CultureInfo.NumberFormatInfo.PercentPositivePattern** and **PercentNegativePattern** properties.

  It defaults to `false`.

- **ShowThousandsSeparator** (Boolean) – When `true`, the thousand separator is added into the textbox as it is reformatted. The thousands separator character is defined in **PeterBlum.DES.Globals.Page.CultureInfo**. When `false`, no thousand characters are shown. If the user had entered them, they would be removed. It defaults to `false`.

  Note: This TextBox always permits entry of the thousands separator regardless of this property.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Spinner Properties

The Spinner is an extension to the PercentTextBox. It provides a pair of arrow buttons that increment or decrement the value of the textbox when clicked. You can customize the button appearance and autorepeat speed with properties on **PeterBlum.DES.Globals.Page.SpinnerManager**. It respects the limits established with the **MinValue** and **MaxValue** properties.

The following properties are identified in the category "Spinner" under the Properties Editor:

- **ShowSpinner** (Boolean) – When `true`, the spinner control is shown. When `false`, it is not. It defaults to `false`.
- **IncrementValue** (Double) – The number to add or subtract to the current value. It supports decimal values. It defaults to 1.0.
- **ContainerMode** (enum PeterBlum.DES.ContainerMode) – When the spinner is shown, the HTML output looks like this:

  ```
  <input type='text' [attributes]/><table>[spinner buttons]<table/>
  ```

  When you assign absolute positioning ("gridlayout"), the style attribute used for absolute positioning would normally get assigned to the `<input>` tag, only affecting its position. If nothing else was done, you would see the textbox in the correct location while the buttons would be elsewhere on the page.

  In addition, these tags can wrap around, especially when placed in a `<p>` tag or a table cell that is too small. The spinners appear below the textbox when that happens.

  Use the **ContainerMode** property to add either a `<span>` or `<table>` tag around this control that limits wrap around. This property is only used when you are showing spinners. When using absolute positioning, the styles used for positioning are moved into the container tag so all of this control's tags are grouped under a common position.

  The enumerated type `PeterBlum.DES.ContainerMode` has these values:

  - `None` – Do not create a container. Use it when you want to create your own container or otherwise control the formatting. If you need to absolutely position this control, you must assign the styles for absolute positioning to your own container.
  - `Auto` – Creates either a `<span>` or `<table>` depending on the browser's support of these styles: `white-space:nowrap` (used by `<span>`) and `display:inline-block` (used by `<table>`). Internet Explorer 5+ and the Mozilla-based browsers support `white-space:nowrap` (as determined by the **PeterBlum.DES.Globals.Page.Browser.MakeWhiteSpaceNoWrap** property.) Opera 7+ and Safari support `display:inline-block` (as determined by the **PeterBlum.DES.Globals.Page.Browser.MakeBlockInlineValue** property.)

    This is the default.

    Note: A tag generates much smaller HTML than the <table> tag.
  - `Span` – Creates a `<span>` tag. It uses the style `white-space:nowrap` to prevent wrapping. Some browsers do not support this style and it will be omitted, leaving the control susceptible to wrapping on those browsers. (The **PeterBlum.DES.Globals.Page.Browser.MakeWhiteSpaceNoWrap** property determines support for this style.)

    The `<span>` tag will also be assigned the style `vertical-align:text-bottom` to keep all elements in a reasonable layout. Yet, some browsers do not make the alignments for this style as well as Internet Explorer and Mozilla-based browsers.
  - `Table` – Creates a `<table>` around the controls with each HTML element in its own table cell. While this is the most reliable, it has these limitations on browsers other than Internet Explorer for Windows, Mozilla, FireFox, and Netscape 6+:

    Some browsers do not support the `display:inline` or `display:inline-block` style required to position the table inside a row of text. A `<span>` tag will be generated instead.

    It uses the style `vertical-align:middle` to attempt to align all elements. Yet, some browsers do not make the alignments for this style as well as Internet Explorer and Mozilla-based browsers.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

It sets `padding` and `margin` styles to 0px on all sides of the cells. It sets `cellpadding` and `cellspacing` to 0 as well. These attempt to remove any shifting imposed by the table's margins. Yet, some browsers do not make the positioning as well as Internet Explorer and Mozilla-based browsers. You may notice misalignment with other elements in the same row.

Note: A tag generates much smaller HTML than the <table> tag.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

FilteredTextBox Control

The FilteredTextBox prevents the user from entering invalid characters into a textbox. You specify the set of characters that are valid or invalid. It does not establish any pattern where specific characters belong. Use a RegexValidator to validate that the characters also match to a pattern. Alternatively, use the MultiSegmentDataEntry control instead of the FilteredTextBox for strongly patterned text.

Click on any of these topics to jump to them:

- Features
- Using the FilteredTextBox
  - Setting the Character Set
  - Getting and Setting the Value of the TextBox
  - When the TextBox is Empty
  - Validation on AutoPostBack
  - Interactive Hints
  - AutoComplete and "Smart Change System"
  - Other Behaviors
- Adding a FilteredTextBox Control
- Properties of the FilteredTextBox

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Features

Use demos here: http://www.peterblum.com/DES/DemoFiltered.aspx.

The `PeterBlum.DES.FilteredTextBox` prevents the user from entering invalid characters into a textbox. You specify the set of characters that are valid or invalid.

It does not apply a pattern to the characters entered. Users can enter any character from the set of characters at any location and for as many times as desired. Typically you will use a RegexValidator or the MultiSegmentDataEntry control to verify that the text matches a pattern.

The CharacterValidator intelligently configures itself to the settings you assign to the FilteredTextBox so that you don't have to do it twice.

Some common uses of this control are:

- Password definition
- Person's name (usually doesn't have punctuation or numbers)
- Phone number
- Credit card number

The `PeterBlum.DES.FilteredTextBox` inherits all of the features of the Enhanced TextBox.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the FilteredTextBox

Use demos here: http://www.peterblum.com/DES/DemoFiltered.aspx.

The `PeterBlum.DES.FilteredTextBox` is subclassed from `PeterBlum.DES.TextBox`, which is an enhanced version of the TextBox control supplied with ASP.NET. If you know how to use the ASP.NET TextBox, you already know how to use this control. See "Using the Enhanced TextBox Control".

Click on any of these topics to jump to them:

- Setting the Character Set

The following topics are inherited from DES's TextBox control:

- Getting and Setting the Value of the TextBox
- When the TextBox is Empty
- Validation on AutoPostBack
- Interactive Hints
- AutoComplete and "Smart Change System"
- Other Behaviors

Setting the Character Set

To establish keystroke filtering, you define the character set that you want to allow or omit. The properties **LettersUppercase**, **LettersLowercase**, **Digits**, and **Space** make it easy to define these common types of characters. Each of these is a Boolean value. Set it to `true` to add to use its characters.

These properties provide some categorized charactersets: **Punctuation**, **CurrencySymbols**, **EnclosureSymbols**, **MathSymbols**, and **VariousSymbols**.

For any other character, or for a subset of the predefined character definitions, add all the desired characters into the **OtherCharacters** property.

For example, if you want all letters, all digits, underscore and period, this is how to set the properties:

```
<des:FilteredTextBox id="FilteredTextBox1" runat="server"
   LettersUppercase="True" LettersLowercase="True" Digits="True"
    OtherCharacters="_." />
```

If you want to omit the set of characters from the field, set the **Exclude** property to `true`.

If you have set the **TextMode** property to `Multiline`, use the **Enter** property to determine if the ENTER key is permitted or not.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Adding a FilteredTextBox Control

These steps ask you to jump around the document using clicks on links. Adobe Reader offers a **Previous View** command to return to the link. Look for this in the Adobe Reader (shown v6.0)

1. Prepare the page for DES controls. See "Preparing a page for DES controls" in the **General Features Guide**. It covers issues like style sheets, AJAX, and localization.
2. Add a FilteredTextBox control to the page.

Visual Studio and Visual Web Developer Design Mode Users

Drag the FilteredTextBox control from the Toolbox onto your web form.

Text Entry Users

Add the control (inside the <form> area):

```
<des:FilteredTextBox id="[YourControlID]" runat="server" />
```

Programmatically creating the FilteredTextBox control

- Identify the control which you will add the FilteredTextBox control to its **Controls** collection. Like all ASP.NET controls, the TextBox can be added to any control that supports child controls, like Panel, User Control, or TableCell. If you want to add it directly to the Page, first add a PlaceHolder at the desired location and use the PlaceHolder.
- Create an instance of the FilteredTextBox control class. The constructor takes no parameters.
- Assign the **ID** property.
- Add the FilteredTextBox control to the **Controls** collection.

In this example, the FilteredTextBox is created with an **ID** of "FilteredTextBox1". It is added to PlaceHolder1.

[C#]

```
PeterBlum.DES.FilteredTextBox vTextBox = new PeterBlum.DES.FilteredTextBox();
vTextBox.ID = "FilteredTextBox1";
PlaceHolder1.Controls.Add(vTextBox);
```

*Note: The namespace for these controls is PeterBlum.DES. If you prefer, add a **using** clause to that namespace on your form.*

[VB]

```
Dim vTextBox As PeterBlum.DES.FilteredTextBox =  _
   New PeterBlum.DES.FilteredTextBox()
vTextBox.ID = "FilteredTextBox1"
PlaceHolder1.Controls.Add(vTextBox)
```

*Note: The namespace for these controls is PeterBlum.DES. If you prefer, add an **Imports** clause to that namespace on your form.*

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Guidelines for setting properties

- Design mode users can use the Properties Editor or the Expanded Properties Editor. (See "Expanded Properties Editor" in the **General Features Guide**.) The SmartTag also offers some of the most important properties.
- Text entry users should add the properties into the `<des:ControlClass>` tag in this format: `propertyname="value"`
- When setting a property programmatically, have a reference to the control's object and set the property according to your language's rules.

3. Set the properties associated with the FilteredTextBox. See "Properties of the FilteredTextBox".
4. Assign Validators to the TextBox. At minimum, assign the CharacterValidator because when JavaScript is not available, the user can enter anything. At runtime, it will automatically configure its properties to match those on the FilteredTextBox. If you need to validate a pattern, use the RegexValidator instead of the CharacterValidator. See the **Validators User's Guide**.

Using DES Validation Framework

See the **Validation User's Guide** for details on these validators.

- Always add one of these validators to block formatting errors because when JavaScript is not available, the user can enter anything:
  - Use the CharacterValidator when there is no pattern to the text. At runtime, it will automatically configure its properties to match those on the FilteredTextBox.
  - Use the RegexValidator when there is a pattern to your text. Make sure your regular expression prevents illegal characters.
- Is this data a candidate for a SQL Injection or Cross Site Scripting attack? Use the PageSecurityValidator or FieldSecurityValidator. See **Input Security User's Guide** for details.
- **Always** set up server side validation. Test **PeterBlum.DES.Globals.Page.IsValid** in your postback event handler methods. Only use the data if it is `true`.

Using Native Validation Framework

- Always add the RegularExpressionValidator to block formatting errors because when JavaScript is not available, the user can enter anything.

  See http://developer.mozilla.org/en/docs/Core_JavaScript_1.5_Reference:Global_Objects:RegExp for regular expression assistance.

  When there is no pattern to the text, here is the usual regular expression pattern.

  ```
  ^[legal characters here]*$
  ```

  ```
  ^[^illegal characters here]*$
  ```

  See Examples.

  When there is a pattern to your text, see this site for assistance: http://regexlib.com.
- Is this data a candidate for a SQL Injection or Cross Site Scripting attack? Build server side defenses to neutralize the attack. See **Input Security User's Guide** for details.
- **Always** set up server side validation. Test **Page.IsValid** in your postback event handler methods. Only use the data if it is `true`.

5. Get and set the value using the **Text** property.

   Usually you will set the initial value in the `Page_Load()` method when **Page.IsPostBack** is `false`. You will get the submitted value in your post back event method, after checking that **PeterBlum.DES.Globals.Page.IsValid** (DES validation framework) or **Page.IsValid** (native validation framework) is `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

6. Here are some other considerations:

- If you are using an AJAX system to update this control, set the **InAJAXUpdate** property to `true`. Also make sure the PageManager control or AJAXManager object has been setup for AJAX. See "Using these Controls With AJAX" in the **General Features Guide**. *Failure to follow these directions can result in incorrect behavior and javascript errors.*
- This control does not preserve most of its properties in the ViewState, to limit its impact on the page. If you need to use the ViewState to retain the value of a property, see "The ViewState and Preserving Properties for PostBack" in the **General Features Guide**.
- If you encounter errors, see the "Troubleshooting" section for extensive topics based on several years of tech support's experience with customers.
- See also "Additional Topics for Using These Controls".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties of the FilteredTextBox

Most of the properties are inherited from `PeterBlum.DES.TextBox` (see "Properties for the Enhanced TextBox") and the ASP.NET TextBox (see "System.Web.UI.WebControls.TextBox Members").

Click on any of these topics to jump to them:

- Get and Set The Value Properties
- Character Set Rules Properties

The following topics are inherited from DES's TextBox control:

- Editing Properties
- Behavior Properties
- Appearance Properties
- AutoPostBack Properties
- Value When Blank Properties
- ToolTip Properties
- Hint Properties
- Tab Rules Properties
- Client-Side Functions Properties

Get and Set The Value Properties

The Properties Editor shows these properties in the "Data" category.

- **Text** (string) – Get and set the value that is edited. No validation is performed when getting or setting the value. It defaults to "".

  Usually you will set the initial value in the `Page_Load()` method when **Page.IsPostBack** is `false`. You will get the submitted value in your post back event method, after checking that **PeterBlum.DES.Globals.Page.IsValid** is `true`.

- **IsValid** (Boolean) – When `true`, the value of **Text** matches the character set rules. It also returns `true` when blank. Recommendation: Use a CharacterValidator or RegexValidator to validate the value.

- **TextChanged** (event) – This event is fired on post back when the TextBox value has changed. See System.Web.UI.WebControls.TextBox.TextChanged Event.

  Note: This event only works when the ViewState is enabled on the TextBox.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Character Set Rules Properties

The Properties Editor shows these properties in the "Character set" category.

- **LettersLowercase** (Boolean) – When `true`, all lowercase letters are part of the character set. When `false`, no lowercase letters are included unless they are specified in the **OtherCharacters** property. It defaults to `false`.
- **LettersUppercase** (Boolean) – When `true`, all uppercase letters are part of the character set. When `false`, no uppercase letters are included unless they are specified in the **OtherCharacters** property. It defaults to `false`.
- **DiacriticLetters** (Boolean) – When `true`, diacritic (accented) letters are part of the characterset. There are two groups of diacritics available to users: ASCII and Unicode. They are both supported.

  When `false`, no diacritic letters are included unless they are specified in the **OtherCharacters** property. It defaults to `false`.

  ASCII Diacritic Letters Supported

  `âäàåáÄÃ çÇ éêëèÉ æÆ ƒ ïîìí ñÑ ôöòóÖ üûùúÜ ÿ`

  These letters are found in the ASCII character set between ASCII 128 and 165. Users on Windows type ALT+1## to insert them into the textbox. Users on MacOSX use **Edit; Special Characters** to open the Character Palette. These characters are in the Glyph View.

  Unicode Diacritic Letters Supported

  `àáâãäåÀÁÂÃÄÅ æÆ çÇ èéêëÈÉÊË ìíîïÌÍÎÏ ñÑ òóôõöøÒÓÔÕÖØ ùúûüÙÚÛÜ ýÿÝ ðÐ þÞ ß`

  These letters are found in Unicode's Latin-1 Supplement character set between 0192 (Hex 00C0) and 0255 (Hex 00FF). Users on Windows type ALT+0### to insert them into the textbox. Users on MacOSX use **Edit; Special Characters** to open the Character Palette. These characters are in the Unicode View.

- **Digits** (Boolean) – When `true`, all digits are part of the character set. When `false`, no digits are included unless they are specified in the **OtherCharacters** property. It defaults to `false`.
- **Space** (Boolean) – When `true`, the Space character is part of the character set. It defaults to `false`. The **Trim** property automatically strips leading and trailing spaces when `true`. It will not affect embedded spaces.
- **Enter** (Boolean) – When `true`, the Enter character is part of the character set. It defaults to `false`.
- **Punctuation** (Boolean) – When `true`, punctuation characters are part of the characterset. When `false`, no punctuation characters are included unless they are specified in the **OtherCharacters** property. It defaults to `false`.

  Punctuation Characters Supported

  `. , ! ? ' " - ; :`

  (period, comma, exclamation point, question mark, single quote, double quote, dash, semicolon, colon)

  ALERT: Hackers use the single quote and dash for SQL Injection attacks. If you permit these characters, see the DES: Peter's Input Security User's Guide for defensive measures.

- **CurrencySymbols** (Boolean) – When `true`, currency symbol characters are part of the characterset. When `false`, no currency symbol characters are included unless they are specified in the **OtherCharacters** property. It defaults to `false`.

  Currency Symbols Supported

  `$ ¢ ¥ ¥ £ ¤`

  (US Dollar, cents – UNICODE 0162, Yen – ASCII 157, Yen – UNICODE 0164, Pounds – UNICODE 0163)

- **EnclosureSymbols** (Boolean) – When `true`, characters that enclose (bracket) text are part of the characterset. When `false`, no enclosure characters are included unless they are specified in the **OtherCharacters** property. It defaults to `false`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Enclosure Symbols Supported

`( ) [ ] { }`

- **MathSymbols** (Boolean) – When `true`, math symbol characters are part of the characterset. When `false`, no math symbol characters are included unless they are specified in the **OtherCharacters** property. It defaults to `false`.

  Math Symbols Supported

  `+ - * / = ( ) < > . % ± × ÷ ± ÷`

  (plus, minus, asterisk, equals, left paren, right paren, less than, greater than, period, percent, plus/minus – UNICODE 0177, multiply – UNICODE 0215, divide – UNICODE - 0247, divide – ASCII 246)

  ALERT: Hackers use the dash for SQL Injection attacks. If you permit the dash, see the DES: Peter's Input Security User's Guide for defensive measures.

- **VariousSymbols** (Boolean) – When `true`, various symbol characters shown below are part of the characterset. When `false`, none of the characters shown below are included unless they are specified in the **OtherCharacters** property. It defaults to `false`.

  Various Symbols Supported

  `_ @ # ^ & * ~ ¿ ¿ ¡ \ / | ¦ § © ® ` ´`

  (Underscore, at, circumflex, ampersand, asterisk, tilde, inverted question mark – ASCII 168, inverted question mark – UNICODE 0191, inverted exclamation point – UNICODE 0161, left slash, right slash, pipe, broken bar, section sign, copyright, registered, grave accent, acute accent)

- **OtherCharacters** (string) – Enter each unique character that you want in the character set. Don't enter any characters that are already covered by the previously stated properties. Suppose you want to support only "n", "x", and the underscore character here. You would enter "`nx_`". Suppose you want the punctuation characters. You would enter "`.!,?;:`".

  It defaults to "".

- **Exclude** (Boolean) – When `false`, the character set represents only the characters considered valid. When `true`, the character set represents the invalid characters and all other characters are valid. It defaults to `false`.

- **UseKeyboardFiltering** (Boolean) – Determines if client-side filtering is on or off. While the FilteredTextBox is assumed to be filtering and this starts as `true`, some applications need to programmatically determine if filtering is used. They may prefer the Enhanced TextBox at times and when they do, use a FilteredTextBox and switch this to `false`.

  When false, all character set properties are ignored and every character is permitted. In addition, the CharacterValidator assigned to the FilteredTextBox

  It defaults to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

MultiSegmentDataEntry Control

The `PeterBlum.DES.MultiSegmentDataEntry` control is for strongly patterned text entry where the value has data that the user should enter and formatting that guides the user. A U.S. phone number is a good example. It is ten digits with separators after the first 3 and second 3 digits. For example: 123 456 6789 and (123)456-6789. Other common patterns include IP addresses, postal codes, serial numbers, identification numbers like the U.S. social security number, bar code numbers, and dates.

The patterns can include letters as well as digits. For example, a person's first and last name can be entered into two fields although stored in a single text string with a space separating the two.

The MultiSegmentDataEntry control is similar to a masked text box that you often find in Windows applications. It allows the user to type a limited character set into specific segments while imposing specific separators. It's intended to closely assist the user so that they enter data in the correct pattern. The MultiSegmentDataEntry control has some advantages over the masked text box that assist the user further, including segments with dropdownlists, textboxes with spinners, interactive hints on individual segments, and Validators supported both on the entire field and individual segments.

Click on any of these topics to jump to them:

- Features
- Using the MultiSegmentDataEntry Control
  - Getting and Setting the Value of the Control
  - Validation on AutoPostBack
  - Interactive Hints
  - Data Entry Rules
  - Changing the Appearance with Style Sheets
- Adding a MultiSegmentDataEntry Control
- Properties for the MultiSegmentDataEntry Control
- Properties for the PeterBlum.DES.TextSegment Class
- Properties for the PeterBlum.DES.IntegerTextSegment Class
- Properties for the PeterBlum.DES.DropDownListSegment Class
- Examples
- Subclassing MultiSegmentDataEntry
- MultiSegmentDataEntryValidator

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Features

Use demos here: http://www.peterblum.com/DES/DemoMultiSeg.aspx.

Use the `PeterBlum.DES.MultiSegmentDataEntry` control as a substitute for a TextBox when you have a strongly patterned data type. It is a similar idea to a masked textbox, where each character position requires a specific character. For example, this control and masked textboxes are used to enter phone numbers, IP addresses, and dates (although the **Peter's Date And Time** module provides much better date entry with its own DateTextBox.)

While the masked textbox is one TextBox control with precise keyboard filtering, the MultiSegmentDataEntry control defines multiple TextBoxes or DropDownLists, for each "segment" of the data where the user types. Any static text, like the period found between each segment of an IP address, is displayed between the segments and the user doesn't have to type them. This design has several advantages over the masked textbox:

- Browsers have a mixture of capabilities when it comes to handling typing at a particular position. To do it right, you need to know the start and end index of the insertion point (when they are different, they user has selected some text). Internet Explorer does not make this information available, although the Mozilla browsers do. Usually masked textboxes for Internet Explorer can allow illegal cases as the user moves the insertion point within the existing text. The MultiSegmentDataEntry control never has this problem. It does not need to know about the insertion point.
- Each segment's textbox can have its own character set. For example, one can allow letters while another allows digits.
- Segments can offer a DropDownList, which is a very good user interface for having a limited set of choices, like the months of the year.
- Individual segments know when you type a character that separates two segments, like the period between IP address segments. They autotab to the next segment so the user can enter the text naturally, without worrying about the tab key.
- TextBoxes can have a maximum length that provides additional guidance to the user. Plus they can autotab when the limit is hit.
- When working with integers, your textbox can offer spinners (up/down arrows) to change the value.
- Each segment can have its own Validator in addition to a master Validator for the entire text. For example, an IP address needs a RangeValidator for values from 0 to 255 on each segment.
- Hints can be shown on the page as focus moves into a segment. So on-screen documentation is available.

All of these features help greatly improve the user's experience so the user knows what to do and understands how to enter patterned data without knowing the pattern in advance.

To make it work, the MultiSegmentDataEntry control has the ability to get and set single patterned string, splitting or joining it according to rules that you specify. For example, on a phone number, the minus character is just formatting and the digits found before and after a minus appear in different segments.

The MultiSegmentDataEntry control supports most of the existing Validators that evaluate textboxes. For example, if you set it up for credit card numbers, you can use the CreditCardNumberValidator on it. Additionally, since each segment has rules like text length, valid characters, and "requires text", the MultiSegmentDataEntryValidator validates any pattern.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the MultiSegmentDataEntry Control

Use demos here: http://www.peterblum.com/DES/DemoMultiSeg.aspx.

To decide if the MultiSegmentDataEntry control is suitable, you must determine if you have a single piece of data that has a strong pattern. Typically a single piece of data is represented in one string. You will determine the number of segments and their characteristics, including character set, presentation, and characters that users often type to separate each segment. The MultiSegmentDataEntry control lets you assign the entire string and it will split it into individual segments. On post back, you can retrieve a single string, completely formatted in the way you prefer to store it.

You could also use data types like decimal values (two segments with a decimal character separator) and dates (three segments with a date separator). In this case, you would convert the value associated with an individual segment into text and assign it to the segment. For example, convert the DateTime.Month value to a string and assign it to the "month" segment. On post back, you retrieve the text from each segment and convert it back to your data type.

Initially the MultiSegmentDataEntry control has no segments defined. You create segment objects and add them to the **Segments** property, which is a collection. There are three segment classes defined (you can create your own classes too):

- `PeterBlum.DES.TextSegment` – Supplies a `PeterBlum.DES.FilteredTextBox` and rules for defining the character set.
- `PeterBlum.DES.IntegerTextSegment` – Supplies a `PeterBlum.DES.IntegerTextBox`. It only supports entry of integers but it includes two useful tools: spinners and fill the text with lead zeros (where "1" can be shown as "001")
- `PeterBlum.DES.DropDownListSegment` – Supplies a DropDownList control. Use it when you have a short list of choices. In addition, the DropDownList can have an internal value that differs from the name it shows. That value is the actual data. For example, if you make a DropDownList of month names, the values can be the month numbers.

You can use any number of segments and mixture of these classes.

Click on any of these topics to jump to them:

- Defining Segments
  - Segment Validation Rules
  - Splitting and Joining Rules
  - Formatting The Segment
- Getting and Setting the Value of the Control
  - Data Entry Validation
- Validation on AutoPostBack
- Interactive Hints
- Data Entry Rules
- Changing the Appearance with Style Sheets

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Defining Segments

Each segment class defines extensive properties to assist in validation rules, how to split and join a string, what formatting appears before and after the data entry field, separator characters, and characters for other situations.

Segment Validation Rules

Validation rules include the character set (**LettersUppercase**, **LettersLowercase**, **Digits**, **OtherCharacters**), the minimum and maximum lengths (**MinLength**, **MaxLength**), and if an entry is required (**Required**). For the TextSegment, the character set also defines keystroke filtering. These rules alone cannot report an error to the user. That's the job of the MultiSegmentDataEntryValidator which does its work based on these properties.

When an entry is not required, you can autofill a blank field with the value of **TextWhenBlank**.

Splitting and Joining Rules

When working with a string as your data, you can let the MultiSegmentDataEntry control split and join it between the segments. Each segment defines the character sets of the editable data and separators surrounding that editable data. For example, with a phone number of this format `(###) ###-####`, the first segment allows three digits. It is surrounded by the parentheses characters.

The Validation Rules have already provided properties to define the character set and maximum size of the editable data: **LettersUppercase**, **LettersLowercase**, **Digits**, **OtherCharacters**, and **MaxLength**. (IntegerTextSegment is limited to digits and does not offer the character set properties.)

There are two sources for separators. **FormattingTextBefore** and **FormattingTextAfter** are the characters normally found separating the segment. When rejoining the segments into one string, they are always inserted around the data from each segment. For example, the phone number pattern `(###) ###-####` uses "`(`" in **FormattingTextBefore** and "`) `" in **FormattingTextAfter**.

IgnoreTheseCharsBefore and **IgnoreTheseCharsAfter** let you define characters that may appear in the original string but do not follow the format defined on this MultiSegmentDataEntry control. They help avoid splitting errors by ignoring unneeded but expected characters. For example, the phone number may have additional spaces and alternative separators that you want to ignore. This example has spaces inside the parenthesis and a period separating the second and third segments when a dash is the desired format: `( ### ) ###.####`. It would use **IgnoreTheseCharsBefore** = "[space]". If **FormattingTextAfter** does not include a space, **IgnoreTheseCharsAfter** = "[space]" too.

Formatting The Segment

Use these properties to display text before and after the data entry control: **DisplayTextBefore** and **DisplayTextAfter**. They can be any HTML you like. For example, if a space is normally separating two segments, you can use "` `". You can even create labels.

All segments provide these formatting properties: **CssClass** (for style sheets), **Tooltip**, **Hint** (uses the Hint system), and **Width**.

Textbox segments all provide some properties common to `PeterBlum.DES.TextBox`: **TextAlign**, **TabOnTheseKeys**, **DisableAutoComplete**, and **DisablePaste**.

IntegerTextBox segments provide some properties common to `PeterBlum.DES.IntegerTextBox`: **ShowSpinner** (and spinner related properties), **AllowNegatives**, and **FillLeadZeros**.

DropDownList segments provide some properties common to a DropDownList control, used to set up its list: **Items**, **DataSource**, **DataMember**, **DataTextField**, **DataValueField**, and **DataTextFormatString**.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Getting and Setting the Value of the Control

Once you have defined your segments, you can start using the MultiSegmentDataEntry control.

To get and set a string value, use the **Text** or **TextNoSeparators** properties. Both know how to split your string using the rules you have defined on each segment. When getting the string from **TextNoSeparators**, it omits the separators defined in **FormattingTextBefore** and **FormattingTextAfter**.

Use **IsValid** to determine if all segments match their validation rules. Use **IsEmpty** to determine if all segments are empty.

Data Entry Validation

Consider validation a manditory part of data entry. It prevents illegal entries from getting into your database. Also protect yourself by setting up server side validation. Hackers often turn off javascript in their browser in hopes that your server side code doesn't protect against their illegal data.

Note: Normally a Validator will fire as an individual textbox is changed. When you have a group of interconnected textboxes, its better to delay this until focus is no longer in the control. MultiSegmentDataEntry control handles this automatically.

DES Validation Framework Guidelines

See the **Validation User's Guide** for details on these validators.

- Always add a validator that confirms the entry matches your data requirements.
    - The MultiSegmentDataTypeValidator always matches the validation rules you already defined on each segment. See "Segment Validation Rules".
    - Sometimes you have a pattern that is fully covered by the segment validation rules. Define a regular expression for your pattern and use it with the RegexValidator. Here is a great site for popular patterns: http://regexlib.com.
- The MultiSegmentDataEntry control works with DES validators that evaluate text, such as RequiredTextValidator, RegexValidator, DataTypeCheckValidator, or CreditCardNumberValidator as needed. Some of Validators need the formatting separators, like DataTypeCheckValidator. Some do not, like CreditCardNumberValidator. Use the **ValidatorsUse** property to select what the Validator will see.
- You can assign validators to individual segments. For example, establish a range with a RangeValidator or require a pattern with the RegexValidator.

  To attach a Validator to an individual segment, set its **ControlIDToEvaluate** property to the segment's **ID** property programmatically (example uses the 3rd segment):

  [C#]

  ```
  Validator.ControlIDToEvaluate = MultiSegmentDataEntry1.Segments[2].ID
  ```

  [VB]

  ```
  Validator.ControlIDToEvaluate = MultiSegmentDataEntry1.Segments(2).ID
  ```

  Alternatively, set **ControlIDToEvaluate** to the **ID** of the MultiSegmentDataEntry control + "_" + the segment number, starting at 1. If you want to use the 3rd segment on a control whose ID is "PhoneNumber", ControlIDToEvaluate = "PhoneNumber_3".
- Is this data a candidate for a SQL Injection or Cross Site Scripting attack? Use the PageSecurityValidator or FieldSecurityValidator. See **Input Security User's Guide** for details.

Always set up server side validation. Test **PeterBlum.DES.Globals.Page.IsValid** in your postback event handler methods. Only use the data if it is `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Native Validation Framework Guidelines

- Always add a validator that confirms the entry matches your data requirements.
    - The MultiSegmentDataTypeValidator from **PeterBlum.DES.NativeValidators.dll** always matches the validation rules you already defined on each segment. See "Segment Validation Rules".
    - Sometimes you have a pattern that is fully covered by the segment validation rules. Define a regular expression for your pattern and use it with the RegularExpressionValidator. Here is a great site for popular patterns: http://regexlib.com.
- The MultiSegmentDataEntry control works with native validators that evaluate text but only in server side validation. Set the **EnableClientScript** property to `false` on each of these validators.
- You can assign validators to individual segments. For example, establish a range with a RangeValidator or require a pattern with the RegularExpressionValidator. *In this case, client-side validation is supported.*

  To attach a Validator to an individual segment, set its **ControlIDToEvaluate** property to the segment's **ID** property programmatically (example uses the 3rd segment):

  [C#]

  ```
  Validator.ControlIDToEvaluate = MultiSegmentDataEntry1.Segments[2].ID
  ```

  [VB]

  ```
  Validator.ControlIDToEvaluate = MultiSegmentDataEntry1.Segments(2).ID
  ```

  Alternatively, set **ControlIDToEvaluate** to the **ID** of the MultiSegmentDataEntry control + "_" + the segment number, starting at 1. If you want to use the 3rd segment on a control whose ID is "PhoneNumber", ControlIDToEvaluate = "PhoneNumber_3".
- Is this data a candidate for a SQL Injection or Cross Site Scripting attack? Build server side defenses to neutralize the attack. See **Input Security User's Guide** for details.

Always set up server side validation. Test **Page.IsValid** in your postback event handler methods. Only use the data if it is `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Validation on AutoPostBack

If you use **AutoPostBack**, it now automatically validates before posting back except when **AutoPostBackValidates** is `false`. This avoids posting back when there is a validation error.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Interactive Hints

Note: Requires a license for the Peter's Interactive Pages module.

You can show a hint on the page as the user tabs into the textbox. A hint is similar to a tooltip. However, a tooltip only appears if the mouse is over the control. That is not the best way to communicate to the user as they are working in a textbox.

See the "Interactive Hints" section of the **Interactive Pages User's Guide** for details.

Assign your hint text to the **Hint** property on each Segment. If you are using a PopupView, it optionally offers a Help button which can show additional text. That additional text is assigned to the Segment's **HintHelp** property.

The format of hints is determined by a `PeterBlum.DES.HintFormatter` object. You can either define one specific to this control in the **LocalHintFormatter** property or specify the name of one shared by other controls in the **SharedHintFormatterName** property. The HintFormatter determines where the hint is shown:

- PopupView or Label control. A PopupView is similar to a ToolTip, created with HTML and Javascript to float near the control. It can be dragged and closed. It can be customized with style sheets, images, and settings using the **Global Settings Editor**.
- In a tooltip
- In the browser's status bar

Most of the work is done by creating a HintFormatter object. The `PeterBlum.DES.HintFormatter` class describes how the hint text will be displayed. It provides its name, display mode - on the page or in a PopupView, if it's also in the tooltip and/or status bar, and more.

The HintManager object provides many page level properties that support this feature. (**HintManager** is a property of **PeterBlum.DES.Globals.Page** and the PageManager control.) See the "Interactive Hints" section of the **Interactive Pages User's Guide** for details.

Setting Up Hints with PopupViews

1. Review the available PopupView definitions. PopupView definitions are created and edited within the "PopupView definitions for Hints" section of the **Global Settings Editor**. Each has a name, style sheets, images, width, and other behaviors.
2. When using PopupViews, you generally predefine a few HintFormatters that reflect the look and size you need for this page. They are defined in the **HintManager.SharedHintFormatters** property. See the "Interactive Hints" section of the **Interactive Pages User's Guide** for details. If you prefer a control-specific HintFormatter, use the **LocalHintFormatter** property on the TextBox control.
3. Set the **SharedHintFormatterName** property. When using **HintManager.SharedHintFormatters**, it should be the name of the HintFormatter object, the name of a hint PopupView defined in the **Global Settings Editor**, or "{DEFAULT}" to use your global default. Otherwise it should be "".
4. Each HintFormatter object should have its **DisplayMode** property set to `Popup` and **PopupViewName** property set to the desired PopupView name. *When SharedHintFormatterName contains the name of a hint PopupView definition, DES creates a HintFormatter for you with DisplayMode and PopupViewName correctly set.*
5. If you also want to show validation error messages (from the DES Validation Framework) in the PopupView, use the **HintManager.HintsShowErrors** property. See the "Interactive Hints" section of the **Interactive Pages User's Guide** for details.
6. Set the text of the hint in the **Hint** property on each Segment. It can contain HTML tags if desired. If you are using the same text in the **ToolTip** property, you do not need to assign anything to **Hint**. It uses the **ToolTip** property when **Hint** is "".
7. If you are using the **HintFormatter.HelpBehavior** property, set the Segment's **HintHelp** property to the appropriate text, whether it is a more detailed description, a title, a URL, or a script.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Setting Up Hints in a Label or Panel

1. Determine what kind of appearance that you want for your hint. It can be simply a Label or a Panel whose formatting encloses a Label and is fully hidden when there is no hint text to show. See the previous topic.
2. Determine the locations for hints. You can have one on the page, one for each group of controls, or even one for each control. When you put one next to a control, it can be located where Validators appear as there is a feature to prevent the hint from showing when a Validator is shown.
3. Add the controls for hints to the page. Remember that they will be hidden until focus is set to them.
4. If you are using a Panel that contains a Label, make sure the Label's ID is Panel.ID + "_Text".
5. Determine whether you need the HintFormatter object for this control or one can be shared amongst several controls. When using one specific to this control set up the HintFormatter object using the **LocalHintFormatter** property.
6. Otherwise, create the HintFormatter object in the **HintManager.SharedHintFormatters** property.
7. Set the **SharedHintFormatterName** property. When using **HintManager.SharedHintFormatters**, it should be the name of the HintFormatter object or "{DEFAULT}" to use your global default. Otherwise it should be "".
8. Assign the Panel or Label control to the property **HintFormatter.HintControlID**.
9. Set the **HintFormatter.DisplayMode** to `Static` or `Dynamic`. `Static` will preserve the space of the Panel or Label when it is hidden. `Dynamic` will not use any space when hidden.
10. If the Panel or Label appears at the same location as some Validators for this textbox, set **HintFormatter.HiddenOnError** to `true`.
11. If you also want the hint text to appear in the status bar, set **HintFormatter.InStatusBar** to `true`.
12. Set the text of the hint in the **Hint** property on each segment. It can contain HTML tags if desired. If you are using the same text in the **ToolTip** property, you do not need to assign anything to **Hint**. It uses the **ToolTip** property when **Hint** is "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Data Entry Rules

As the user types, the focus jumps from segment to segment based on these properties: **TabAtMaxLength**, **TabOnEnterKey**, **TabByArrowKeys**, and **TabOnBackspace**.

You can make the ENTER key click a button by setting the button's control ID in the **EnterSubmitsControlID** property.

Note: EnterSubmitsControlID requires a license for the Peter's Interactive Pages.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Changing the Appearance with Style Sheets

Initially, this control has the appearance of series of separate data entry controls. Use style sheets to give it a different appearance. Here are two versions of the same control, setup to be a phone number. The second has style sheets applied:

DES supplies style sheets to make this easy. They can quickly be applied using the **Auto Format** command in the SmartTag (▶). Just be aware that it immediately updates the **CssClass** properties on the control and its segments as you make choices.

To establish the outer frame, create a style sheet class and assign its name to the **CssClass** property. DES supplies this class for you in its **DES\Appearance\TextBoxes\TextBoxes.css** file:

```
.DESTBMultiSegContainer
{
   border-left: #d3d3d3 thin inset; /* lightgrey */
   border-top: #d3d3d3 thin inset;
   border-right: #d3d3d3 thin inset;
   border-bottom: #d3d3d3 thin inset;
   padding-left: 2px;
   padding-top: 2px;
   padding-right: 2px;
   padding-bottom: 2px;
}
```

It is used in the second graphic above.

To change the frames of textboxes, create a style sheet class with light borders or borders that match your background and assign it to each segment's **CssClass** property. DES supplies this class for you in its **DES\Appearance\TextBoxes\TextBoxes.css** file:

```
.DESTBMultiSegTextBox
{
   border-left: #f5f5f5 thin solid; /* whitesmoke */
   border-top: #f5f5f5 thin solid;
   border-right: #f5f5f5 thin solid;
   border-bottom: #f5f5f5 thin solid;
}
```

It is used in the second graphic above.

To change the appearance of DropDownLists, create a style sheet class and assign it to each DropDownListSegment's **CssClass** property. DES supplies this class for you in its **DES\Appearance\TextBoxes\TextBoxes.css** file:

```
.DESTBMultiSegDropDownList
{
}
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Adding a MultiSegmentDataEntry Control

These steps ask you to jump around the document using clicks on links. Adobe Reader offers a **Previous View** command to return to the link. Look for this in the Adobe Reader (shown v6.0)

1. Prepare the page for DES controls. See "Preparing a page for DES controls" in the **General Features Guide**. It covers issues like style sheets, AJAX, and localization.
2. Add a MultiSegmentDataEntry control to the page.

Visual Studio and Visual Web Developer Design Mode Users

Drag the MultiSegmentDataEntry control from the Toolbox onto your web form.

Text Entry Users

- Add the control (inside the <form> area)::

```
<des:MultiSegmentDataEntry id="[YourControlID]" runat="server" />
```

Programmatically creating the MultiSegmentDataEntry control

- Identify the control which you will add the MultiSegmentDataEntry control to its **Controls** collection. Like all ASP.NET controls, the MultiSegmentDataEntry can be added to any control that supports child controls, like Panel, User Control, or TableCell. If you want to add it directly to the Page, first add a PlaceHolder at the desired location and use the PlaceHolder.
- Create an instance of the MultiSegmentDataEntry control class. The constructor takes no parameters.
- Assign the **ID** property.
- Add the MultiSegmentDataEntry control to the **Controls** collection.

In this example, the MultiSegmentDataEntry is created with an **ID** of "MultiSegmentDataEntry1". It is added to PlaceHolder1.

[C#]

```
PeterBlum.DES.MultiSegmentDataEntry vMSDE =
   new PeterBlum.DES.MultiSegmentDataEntry();
vMSDE.ID = "MultiSegmentDataEntry1";
PlaceHolder1.Controls.Add(vMSDE);
```

*Note: The namespace for these controls is PeterBlum.DES. If you prefer, add a **using** clause to that namespace on your form.*

[VB]

```
Dim vMSDE As PeterBlum.DES.MultiSegmentDataEntry = _
   New PeterBlum.DES.MultiSegmentDataEntry()
vMSDE.ID = "MultiSegmentDataEntry1"
PlaceHolder1.Controls.Add(vMSDE)
```

*Note: The namespace for these controls is PeterBlum.DES. If you prefer, add an **Imports** clause to that namespace on your form.*

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

3. Add segment objects to the **Segments** property. See "Defining Segments" and "Examples".

Visual Studio and Visual Web Developer Design Mode Users

The Segments property offers this editor.

When you click the **Add** button, a list of segment types appears. Select from TextSegment, IntegerTextSegment, and DropDownListSegment. A new segment will be added. Edit the properties in the right panel.

Text Entry Users

Segments are a type of collection. Therefore its ASP.NET text is nested as a series of child controls within the `<Segments>` tag. Each segment is a tag with `<des:segmentclass>` followed by the properties. For TextSegment, use `<des:TextSegment>`; for IntegerTextSegment, use `<des:IntegerTextSegment>`; for DropDownListSegment, use `<des:DropDownListSegment>`.

The following example represents the same conditions shown in the editor window above.

```
<des:MultiSegmentDataEntry id="PhoneNumber" runat="server">

   <Segments>
      <des:TextSegment CssClass="DESMultiSegTextBox"
         MinLength="3" TextAlign="Center" Width="30px" />
      <des:IntegerTextSegment CssClass="DESMultiSegTextBox"
         FormattingTextAfter="-" TabOnTheseKeys="- "  />
   </Segments>

</des:MultiSegmentDataEntry>
```

Programmatically creating the segment objects

- Create an instance of one of these classes: `PeterBlum.DES.TextSegment`, `PeterBlum.DES.IntegerTextSegment`, or `PeterBlum.DES.DropDownListSegment`. The constructor takes no parameters.
- Assign property values.
- Add the object to the Segments collection using its `Add()` method.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

[C#]

```
PeterBlum.DES.TextSegment vSeg1 = new PeterBlum.DES.TexSegment();
vSeg1.Width = "30px";
vSeg1.MinLength = 3;
vSeg1.TextAlign = PeterBlum.DES.TextAlign;
vSeg1.CssClass = "DESMultiSegTextBox";
PhoneNumber.Segments.Add(vSeg1);
```

[VB]

```
Dim vSeg1 As PeterBlum.DES.TextSegment = New PeterBlum.DES.TexSegment()
vSeg1.Width = "30px"
vSeg1.MinLength = 3
vSeg1.TextAlign = PeterBlum.DES.TextAlign
vSeg1.CssClass = "DESMultiSegTextBox"
PhoneNumber.Segments.Add(vSeg1)
```

Guidelines for setting properties

- Design mode users can use the Properties Editor or the Expanded Properties Editor. (See "Expanded Properties Editor" in the **General Features Guide**.) The SmartTag also offers some of the most important properties.
- Text entry users should add the properties into the `<des:ControlClass>` tag in this format: `propertyname="value"`
- When setting a property programmatically, have a reference to the control's object and set the property according to your language's rules.

4. Set the properties associated with the MultiSegmentDataEntry . See "Properties for the MultiSegmentDataEntry Control".
5. Assign the MultiSegmentDataEntryValidator and other Validators to the MultiSegmentDataEntry control. See "Data Entry Validation".
6. Get and set the value using the **Text** property. If you need to assign values directly to individual segments, convert your value into a string and assign it to the **Text** property on each segment. In this example, an integer in *monthnum* is set on the first segment:

   [C#]

   ```
   StartDate.Segments[0].Text = monthnum.ToString();
   ```

   [VB]

   ```
   StartDate.Segments(0).Text = monthnum.ToString()
   ```

   Usually you will set the initial value in the `Page_Load()` method when **Page.IsPostBack** is `false`. You will get the submitted value in your post back event method, after checking that **PeterBlum.DES.Globals.Page.IsValid** (DES validation framework) or **Page.IsValid** (native validation framework) is `true`.

7. Here are some other considerations:

   - If you are using an AJAX system to update this control, set the **InAJAXUpdate** property to `true`. Also make sure the PageManager control or AJAXManager object has been setup for AJAX. See "Using these Controls With AJAX" in the **General Features Guide**. *Failure to follow these directions can result in incorrect behavior and javascript errors.*
   - This control does not preserve most of its properties in the ViewState, to limit its impact on the page. If you need to use the ViewState to retain the value of a property, see "The ViewState and Preserving Properties for PostBack" in the **General Features Guide**.
   - If you encounter errors, see the "Troubleshooting" section for extensive topics based on several years of tech support's experience with customers.
   - See also "Additional Topics for Using These Controls".

Use demos here: http://www.peterblum.com/DES/DemoMultiSeg.aspx.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties for the MultiSegmentDataEntry Control

The `PeterBlum.DES.MultiSegmentDataEntry` control is subclassed from `System.Web.UI.WebControls.WebControl`. It inherits properties common to all webcontrols.

Click on any of these topics to jump to them:

- Getting and Setting Values Properties
- Segments Property
  - Properties for the PeterBlum.DES.TextSegment Class
  - Properties for the PeterBlum.DES.IntegerTextSegment Class
  - Properties for the PeterBlum.DES.DropDownListSegment Class
- Behavior Properties
- Tab Rules Properties
- Appearance Properties
- Hint Properties

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Getting and Setting Values Properties

None of these properties are shown in the Properties Editor. They must be used programmatically or with databinding.

- **Text** (string) - Gets and sets the text that is used by the segments.

  When getting, it joins the text from each segment to build your string. It retrieves the data from each segment enclosed in the text from the **FormattingTextBefore** and **FormattingTextAfter** properties. If the segment's data is blank, the value of the **TextWhenBlank** property is used.

  When setting, it splits your string using rules on each segment. This is one of the most complex features of this control. Text that closely matches the strong pattern with the correct number of characters in each segment and the same separators identified by **FormattingTextBefore** and **FormattingTextAfter** will always work.

  The splitting function basically parses the text from left to right, adding each character permitted by the current character until:

  - It hits one that is not in the segment's character set for the editable data.
  - Defined as separator (in **FormattingTextAfter**, **IgnoreTheseCharactersAfter** or in the next segments **FormattingTextBefore** and **IgnoreTheseCharactersBefore** properties).
  - Reaches the **MaxLength**. This allows a string consisting only of digits to be split at specific places. For example, a social security number has segments of 3, 2, and 4 digits. If you pass "123456789", it will split it like "123" "45" "6789".

  ### Splitting Badly Formatted Data

  If your database has some existing data that does not match the desired format, see if the **IgnoreTheseCharactersBefore** and **IgnoreTheseCharactersAfter** properties will help. Otherwise, expect that the data will split in unusual ways.

  Suppose that you have a US Phone number pattern, which expects this pattern: `(###) ###-####`. Here are some values and how to make them work:

  555 222-1234 – Works when a space character is defined in the first segment's **FormattingTextAfter**, the second segment's **FormattingTextBefore**, the first segment's **IgnoreTheseCharactersAfter** or the second segment's **IgnoreTheseCharactersBefore**.

  555222-1234 – Works when the first segment's **MaxLength** is set to 3.

  555-222-1234 – Works when the first segment's **IgnoreTheseCharactersAfter** contains a minus character.

  555-222123 – If the second segment has established a **MaxLength**, it will split after "222" leaving the 3rd segment with "123".

  (555) 2 22-1234 – With the space inside what would normally be the second segment, expect a bad format where "2" is in the second segment, "22" is in the third, and "1234" to be omitted. If you have set up good Validators and the user submits this page, they will be required to fix the phone number.

- **TextNoSeparators** (string) – Gets and sets the text used in segments. When setting, it works identically to the **Text** property. When getting, it never adds the **FormattingTextBefore** and **FormattingTextAfter** properties on segments although it will use **TextWhenBlank** if needed.

  Suppose you are getting a phone number whose formatted pattern is `(###) ###-####`. The **Text** property returns (555) 222-1234 while **TextNoSeparators** returns 5552221234.

- **ValidatorText** (string) – Gets the text used by Validators attached to this control. It will use the **ValidatorsUse** property to select between the **Text** and **TextNoSeparators** property. You probably will not use it.

- **IsValid** (Boolean) – Determines if all of the segments have data that matches their validation rules. When `true`, they match their rules. When `false`, the do not. Usually you will use the MultiSegmentDataEntryValidator instead of this property to determine validity.

- **IsEmpty** (Boolean) – Determines if all segments have no data. When `true`, they are all empty. When `false`, at least one has text. You can use a RequiredTextValidator to prevent an empty control.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **TextChanged** (event) – This event is fired on post back when the control's value has changed. It is the same method definition as you use with a TextBox. See System.Web.UI.WebControls.TextBox.TextChanged Event.

  This event only works when the ViewState is enabled on the MultiSegmentDataEntry control.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Segments Property

The Properties Editor shows these properties in the "Data" category.

- **Segments** (PeterBlum.DES.SegmentList) – A collection of segment objects. Add any of the segment classes to it: `PeterBlum.DES.TextSegment`, `PeterBlum.DES.IntegerTextSegment`, and `PeterBlum.DES.DropDownListSegment`. See "Defining Segments" for an overview and "Examples".

  Each segment provides extensive formatting properties for the text before, after, and the data entry control itself.

  - See "Properties for the PeterBlum.DES.TextSegment Class"
  - See "Properties for the PeterBlum.DES.IntegerTextSegment Class"
  - See "Properties for the PeterBlum.DES.DropDownListSegment Class"

  You must have one visible segment for this control to be shown. Segments are shown in the order of this collection.

Visual Studio and Visual Web Developer Design Mode Users

The **Segments** property offers this editor.

When you click the **Add** button, a list of segment types appears. Select from TextSegment, IntegerTextSegment, and DropDownListSegment. A new segment will be added. Edit the properties in the right panel.

Text Entry Users

Segments are a type of collection. Therefore its ASP.NET text is nested as a series of child controls within the `<Segments>` tag. Each segment is a tag with `<des:segmentclass>` followed by the properties. For TextSegment, use `<des:TextSegment>`; for IntegerTextSegment, use `<des:IntegerTextSegment>`; for DropDownListSegment, use `<des:DropDownListSegment>`.

The following example represents the same conditions shown in the editor window above.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

```
<des:MultiSegmentDataEntry id="PhoneNumber" runat="server">

   <Segments>
      <des:TextSegment CssClass="DESMultiSegTextBox"
         MinLength="3" TextAlign="Center" Width="30px" />
      <des:IntegerTextSegment CssClass="DESMultiSegTextBox"
         FormattingTextAfter="-" TabOnTheseKeys="- "  />
   </Segments>

</des:MultiSegmentDataEntry>
```

See also "Examples".

Programmatically creating the segment objects

- Create an instance of one of these classes: `PeterBlum.DES.TextSegment`, `PeterBlum.DES.IntegerTextSegment`, or `PeterBlum.DES.DropDownListSegment`. The constructor takes no parameters.
- Assign property values.
- Add the object to the Segments collection using its `Add()` method.

[C#]

```
PeterBlum.DES.TextSegment vSeg1 = new PeterBlum.DES.TexSegment();
vSeg1.Width = "30px";
vSeg1.MinLength = 3;
vSeg1.TextAlign = PeterBlum.DES.TextAlign;
vSeg1.CssClass = "DESMultiSegTextBox";
PhoneNumber.Segments.Add(vSeg1);
```

[VB]

```
Dim vSeg1 As PeterBlum.DES.TextSegment = New PeterBlum.DES.TexSegment()
vSeg1.Width = "30px"
vSeg1.MinLength = 3
vSeg1.TextAlign = PeterBlum.DES.TextAlign
vSeg1.CssClass = "DESMultiSegTextBox"
PhoneNumber.Segments.Add(vSeg1)
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Behavior Properties

- **InAJAXUpdate** (Boolean) – When using AJAX on this page, set this to `true` if the control is involved in an AJAX update. See "Using These Controls with AJAX" in the **General Features Guide**. It defaults to `false`.
- **Visible** (Boolean) – When `false`, no HTML is output. This control is entirely unused. When `true`, the control generates HTML. It defaults to `true`.
- **Enabled** (Boolean) – When `false`, the segments appear disabled and do not accept modifications. When `true`, they are editable. It defaults to `true`.
- **AutoPostBack** (Boolean) – When `true`, a change on the client-side will immediately submit the page. A change is only detected when focus is no longer in any of the segment's data entry controls. Usually you use this with the **TextChanged** event to update some part of the page and redraw it with that change. It defaults to `false`.

  Note: When using Microsoft ASP.NET AJAX or Telerik RadAjax on the page and this control has InAJAXUpdate set to true, this property will use that AJAX framework to make a callback instead of a postback.
- **AutoPostBackValidates** (Boolean) – When `true` and **AutoPostBack** are `true`, before submitting, it validates the control. If there are any errors, it does not submit. This avoids post back when the field has errors. When `false`, auto post back never validates. It defaults to `true`.
- **ValidatorsUse** (enum PeterBlum.DES.ValidatorsUse) - Determines the format of text used by Validators that access this control (except the MultiSegmentDataEntryValidator). For example, the EmailAddressValidator wants formatting like "@" between segments while CreditCardNumberValidator cannot use separator characters.

  The enumerated type `PeterBlum.DES.ValidatorsUse` has these values:

  - `Text` - Use the **Text** property as a source. It includes separators. This is the default.
  - `TextNoSeparators` - Use the **TextNoSeparators** property as a source.
- **EnterSubmitsControlID** (string) – Use this when you want the ENTER key to click a specific button. The browser already has rules for clicking a button when you type ENTER. That button usually has a special frame to identify it to the user. This will override that button.

  Suppose that you have two groups of fields, each with its own submit button. Each control should use this to point to its own submit button.

  Assign the ID of the submit control. It must be assigned to a control in the same or a parent naming container. If the control is in another naming container, use **EnterSubmitsControl**.

  This feature fires the `click()` method on the client-side control. `click()` automatically runs the control's client-side `onclick` event. In the case of a submit control, it submits the page after firing client-side validation. There are a lot of controls that support `click()`, although they vary by browser. In addition to Buttons and ImageButtons, typical cases are hyperlinks, LinkButtons, checkboxes and radiobuttons. However, browsers don't all support the `click()` method on the same control. Here are the differences:

  - Internet Explorer and Opera 7 support it on hyperlinks (and LinkButton) while Mozilla and Safari do not.
  - All support checkboxes and radiobuttons. However, Mozilla always removes the focus from the current field even if you don't set this feature up to move the focus (the focus is gone, not moved)
  - All support Buttons the same way. This is the best choice for a control to click.

  It defaults to "".

  License Note: This property requires a license for the Peter's Interactive Pages.
- **EnterSubmitsControl** (System.Web.UI.Control) – This is an alternative to **EnablerSubmitsControlID**. It has the same features as **EnablerSubmitsControlID**. It is assigned a reference to a control instead of an ID. As a result, it supports controls in any naming container. It must be assigned programmatically.

  When programmatically assigning properties to a TextBox control, if you have access to the submit control object, it is better to assign it here than assign its ID to the **EnablerSubmitsControlID** property because DES operates faster using **EnablerSubmitsControl**.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

License Note: This property requires a license for the Peter's Interactive Pages.

- **ChangeMonitorGroups** (string) – When using the Change Monitor, the group names defined here are marked changed when this control is edited. See "Change Monitor" in the **Interactive Pages User's Guide**.

  The value of "" is a valid group name.

  For a list of group names, use the pipe character as a delimiter. For example: "GroupName1|GroupName2". If one of the groups has the name "", start this string with the pipe character: "|GroupName2".

  Use "*" to indicate all groups apply.

  It defaults to "".

- **OnChangeScript** (string) - Client-side JavaScript code that will be invoked when the control is found to have changed. With many individual data entry controls, use this to detect when the user has left the entire control after a change was made.

  You must provide valid JavaScript that ends in either a semicolon or closing bracket. It cannot contain a function definition.

  Recommendation: Create a function elsewhere on the page and call it within this string. For example: `"MyFunction(parm1);"`.

  If **AutoPostBack** is `true`, it will append its own code to yours. In that case, your code should not be written in a way that will skip code that follows, such as by using a `return` statement unless it does so to prevent the call to `__doPostBack()` from occurring.

  When "", it is not used.

  It defaults to "".

- **ViewStateMgr** (PeterBlum.DES.ViewStateMgr) – Enhances the ViewState on this control to provide more optimal storage and other benefits. Normally, the properties of this control and its segments are not preserved in the ViewState. Just call `ViewStateMgr.TrackProperty("propertyname")` to save the property. Individual segments have a similar method: `TrackPropertyInViewState("propertyname")`.

  For more details on the `PeterBlum.DES.ViewStateMgr` class, see "The ViewState and Preserving Properties for PostBack" in the **Validation User's Guide**.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Tab Rules Properties

The Properties Editor shows these properties in the "Tab Rules" category.

- **TabAtMaxLength** (Boolean) – When `true`, entry into textboxes will automatically tab to the next segment when the segment reaches **MaxLength**. The last segment will not auto-tab. It defaults to `true`.
- **TabOnEnterKey** (Boolean) – When `true`, textbox segments will automatically tab to the next segment when ENTER it typed. The last segment will not auto-tab. It defaults to `false`.

  Note: EnterSubmitsControlID overrides this property.
- **TabByArrowKeys** (Boolean) – Determines if the arrow keys move focus to the next and previous segments when the cursor reaches the text limit.

  Focus will move to the next segment when the user types a right arrow at the end of the current text.

  Focus will move to the previous segment when the user types a left arrow at the start of the current text.

  It defaults to `true`.
- **TabOnBackspace** (Boolean) – Determines if focus will move to the previous segment when the user types a backspace into an empty textbox.

  It defaults to `false`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Appearance Properties

The Properties Editor shows these properties in the "Appearance" category.

- **CssClass** (string) – Establish a style for the container of this control. The container is either a `<span>` or `<div>` tag depending on **EnclosedBy**. A typical style will establish borders. The **DESStyleSheet.css** file supplies the "DESMultiSegContainer" property for this task. See "Changing the Appearance with Style Sheets".
- **Width** (System.Web.UI.WebControls.Unit) – If you set **EnclosedBy** to use a `<div>` tag, consider establishing a width. By default, there is no width.
- **BackColor**, **BorderColor**, **BorderStyle**, **BorderWidth**, **Columns**, **Font**, **ForeColor**, **Height**, and **Style** – These properties are described in System.Web.UI.WebControls.WebControl Members.

  Recommendation: use the **CssClass** property to establish a style sheet instead of these properties.
- **EnclosedBy** (enum PeterBlum.DES.MSDEEnclosedBy) – Determines if this control is enclosed by a `<span>` or `<div>` tag. The enumerated type `PeterBlum.DES.MSDEEnclosedBy` has these values:
  - `SPAN` - Use a `<span>` tag. It allows the control to appear "inline". This is the default.
  - `DIV` - Use a `<div>` tag. When you embed block-type HTML tags into the segment's **DisplayBeforeText** and **DisplayAfterText**, some browsers do not permit them in a `<span>` tag. Block-type HTML tags include but are not limited to `<div>`, `<table>`, `<tr>`, and `<td>`.

  When using the `<div>` enclosure, consider setting the **Width** attribute. Otherwise, it probably will be too wide.
- **TabIndex** (short) – Determine the tab index of the control on the page. The **TabIndex** is applied to the first segment. Each segment that follows adds one to the **TabIndex**. So if you have 4 segments, it will use 4 sequential values, from TabIndex to TabIndex + 3.
- **ToolTip** (string) – When assigned, a tooltip with this text is shown when the user points to the control. It can be overridden on individual segments which have their own **ToolTip**. If you are using the **Hint** feature, it can be used as the hint when the **Hint** property is "". When using the "Enhanced ToolTips" feature, the browser's tooltip will be replaced by a PopupView. See the **Interactive Pages User's Guide**.
- **ToolTipLookupID** (string) – Gets the value for **ToolTip** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Hints. If no match is found OR this is blank, **ToolTip** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".
- **ToolTipUsesPopupViewName** (string) – When using the "Enhanced ToolTips" feature, this determines which PopupView definition is used. For details on Enhanced ToolTips, see the **Interactive Pages User's Guide**.

  Specify the name from the PopupView definition or use the token "{DEFAULT}" to select the name from the global setting **DefaultToolTipPopupViewName**, which is set with the **Global Settings Editor**.

  A PopupView definition describes the name, style sheets, images, behaviors, and size of a PopupView. Use the **Global Settings** Editor to create and edit these PopupView definitions in the "PopupView definitions used by the HintManager" section.

  Tooltips are only converted to PopupViews when **HintManager.EnableToolTipsUsePopupViews** is `True`. (**HintManager** is accessed programmatically through **PeterBlum.DES.Globals.Page** and in the **PageManager** control.)

  Here are the predefined values: `LtYellow-Small`, `LtYellow-Medium`, `LtYellow-Large`, `ToolTip-Small`, `ToolTip-Medium`, and `ToolTip-Large`. All of these are light yellow. Their widths vary from 200px to 600px. Those named "ToolTip" have the callout feature disabled. Those named "LtYellow" have the callout feature enabled.

  It defaults to "`{DEFAULT}`".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Note: When the name is unknown, it also uses the factory default. This allows the software to operate even if a PopupView definition is deleted or renamed.

Note: When the HintManager.ToolTipsAsHints feature is enabled, anything other than "" or "{DEFAULT}" assigned to ToolTipUsesPopupViewName will prevent the ToolTip text from being assigned as a Hint. You must explicitly assign the Hint text if you want the tooltip and hint to share the same text.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Hint Properties

License Note: This feature requires a license for the Peter's Interactive Pages.

The Interactive Hint feature lets you show text on-screen that describes the segment currently with focus. See "Interactive Hints" to learn how to set up the Hint feature.

The actual text of a hint is assigned on individual segments, using their **Hint** and **HintHelp** properties. The rest of the properties, shown below, are on the MultiSegmentDataEntry control.

The Properties Editor shows these properties in the "Hint" category.

- **SharedHintFormatterName** (string) – Specify the name of the desired HintFormatter object found in **HintManager.SharedHintFormatters**. (HintManager is accessed programmatically through **PeterBlum.DES.Globals.Page** and in the **PageManager** control.) Alternatively, specify the name of a PopupView defined in the "PopupView definitions used by HintFormatters" of the **Global Settings Editor**.

  The `PeterBlum.DES.HintFormatter` class describes how the hint text will be displayed. It provides its name, display mode - on the page or in a PopupView, if it's also in the tooltip and/or status bar, and more.

  The **HintManager.SharedHintFormatters** property defines various ways to display a hint with `PeterBlum.DES.HintFormatter` objects. It lets you share a HintFormatter definition amongst controls on this page. It not only makes changes to the HintFormatter quick, but it also reduces the JavaScript output. If you want to create a HintFormatter specific to this control, set **SharedHintFormatterName** to "" and edit the properties of **LocalHintFormatter** (see below).

  If you specify the name of a PopupView and there is a definition with that name, a HintFormatter is automatically added to **HintManager.SharedHintFormatters** with its name matching the name of the PopupView. This is an easy way to work with PopupViews without the extra step of setting up HintFormatters. The HintFormatter defined will also show the hint as a tooltip but it will not show the hint in the status bar. If you need more control over the HintFormatter's properties, you must create the HintFormatter yourself.

  See the "Interactive Hints" section of the **Interactive Pages User's Guide** for details on the `PeterBlum.DES.HintFormatter` class and setting up **HintManager.SharedHintFormatters**.

  Use the token "{DEFAULT}" to get the name from **HintManager.DefaultSharedHintFormatterName**.

  It defaults to "`{DEFAULT}`".

- **LocalHintFormatter** (PeterBlum.DES.HintFormatter) – When none of the HintFormatter objects defined in **HintManager.SharedHintFormatters** is appropriate, use this property. (HintManager is accessed programmatically through **PeterBlum.DES.Globals.Page** and in the **PageManager** control.)

  The `PeterBlum.DES.HintFormatter` class describes how the hint text will be displayed. It provides its display mode - on the page or in a PopupView, if it's also in the tooltip and/or status bar, and more. See the "Interactive Hints" section of the **Interactive Pages User's Guide** for directions on using the `PeterBlum.DES.HintFormatter` class.

  You must set **SharedHintFormatterName** to "" for this to be used.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties for the PeterBlum.DES.TextSegment Class

For directions on creating a `PeterBlum.DES.TextSegment` object, see step 0 in "Adding a MultiSegmentDataEntry Control".

By default, no properties are saved in the control's ViewState. If you set them programmatically, you must do it every time. Also you can save an individual property in the ViewState by calling the `TrackPropertyInViewState("propertyname")` method on the Segment object.

Click on any of these topics to jump to them:

- Get and Set Text Properties
- Split and Join Text Properties
- Data Entry and Validation Rule Properties
- Appearance Properties
- Behavior Properties
- Hint Properties

Get and Set Text Properties

These three properties interact directly with the TextBox's text value. Usually you will assign a string to the **MultiSegmentDataEntry.Text** and **TextNoSeparator** properties and they will split and join the string amongst the segments. See the next topic. Use these when you are working with something other than a string whose elements you convert to text and are associated with individual segments.

- **Text** (string) – Gets and sets the value assigned to the TextBox. It does not validate or parse the text. If you want the MultiSegmentDataEntry control to parse a string, assign your string to **MultiSegmentDataEntry.Text** or **MultiSegmentDataEntry.TextNoSeparator**.

  Use this when MultiSegmentDataEntry does not handle your data format. For example, when editing a date, you may start with a `System.DateTime` structure, not a string. So convert each element – day, month, and year – to a string, then assign it here.

- **IsValid** (Boolean) – Returns `true` when the TextBox matches the segment's validation rules, like **Required**, the character set, and text length. Returns `false`, when it fails to match the validation rules. Usually you will attach a MultiSegmentDataEntryValidator to the MultiSegmentDataEntry control to validate it.

- **IsEmpty** (Boolean) – Returns `true` when the TextBox is blank. Returns `false` when it has text. When the TextBox contains a value matching **TextWhenBlank**, it is not considered empty.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Split and Join Text Properties

These properties support the **MultiSegmentDataEntry.Text** and **TextNoSeparator** properties as they split and join one string amongst the segments.

- **MaxLength** (integer) – The maximum number of characters required in this segment. A maximum is always required. It provides a limit for splitting the text when you set the **MultiSegmentDataEntry.Text** property.

  This value will be assigned to the TextBox's **MaxLength** property imposing a client-side limit as the user types.

  It defaults to 4.

  Note: There is a MinLength property too. It is not used to split and join but it is used for validation. See the text topic.

- **FormattingTextBefore** (string) – Text that appears before the value that goes into the TextBox. It is used to split and join the string you supply in the **MultiSegmentDataEntry.Text** property.

  When splitting text, if this string is found, it is skipped so the text after it will be assigned to the TextBox.

  When joining text, **FormattingTextBefore** precedes the value from the TextBox.

  If **DisplayTextBefore** and **DisplayTextAfter** are both blank, this property is shown on the page before the TextBox.

  It defaults to "".

- **FormattingTextAfter** (string) – Text that appears after the value that goes into the TextBox. It is used to split and join the string you supply in the **MultiSegmentDataEntry.Text** property.

  When splitting text, if this string is found, it is skipped so the text after it will be assigned to the next segment.

  When joining text, **FormattingTextAfter** follows the value from the TextBox.

  If **DisplayTextBefore** and **DisplayTextAfter** are both blank, this property is shown on the page after the TextBox.

  It defaults to "".

- **IgnoreTheseCharsBefore** (Boolean) – When setting the **MultiSegmentDataEntry.Text** property, the control splits your string amongst each segment. Each character in **IgnoreTheseCharsBefore** will be ignored if found preceding the value that is assigned to the TextBox. It helps the MultiSegmentDataEntry control support badly formatted data.

  When splitting, data you retrieve from your database may contain inappropriate characters, due to inconsistent data entry rules. Perhaps another web page or application was used to collect this data and its validation rules did not stop entries that you now consider illegal formats.

  For example, you demand phone numbers to have this format: `(###) ###-####`. But your database contains phone numbers with additional spaces: `( ### ) ### - ####`. The space character is inappropriate here and can cause the MultiSegmentDataEntry control to parse incorrectly. By adding the space character to **IgnoreTheseCharsBefore**, it will be skipped as the data is split. In fact, the space character is a very common entry for this property.

  This property permits a list of characters. For example, if you want to ignore dash, exclamation point, and slash, just enter `"-!/"`.

  It does a case insensitive match in case you want to include letters here.

  If you enter the text `"a-z"` (in lowercase), it is a special symbol that means consider all letters (including Unicode) to be ignored. Only use it when this segment and the next visible segment do not allow letters in their TextBoxes.

  If you enter the text `"0-9"`, it is a special symbol that means consider all digits to be ignored. Only use it when this segment and the next visible segment do not allow digits in their TextBoxes.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **IgnoreTheseCharsAfter** (Boolean) – When setting the **MultiSegmentDataEntry.Text** property, the control splits your string amongst each segment. Each character in **IgnoreTheseCharsAfter** will be ignored if found after the value that is assigned to the TextBox. It helps the MultiSegmentDataEntry control support badly formatted data.

  When splitting, data you retrieve from your database may contain inappropriate characters, due to inconsistent data entry rules. Perhaps another web page or application was used to collect this data and its validation rules did not stop entries that you now consider illegal formats.

  For example, you demand phone numbers to have this format: `(###) ###-####`. But your database contains phone numbers with additional spaces: `( ### ) ### - ####`. The space character is inappropriate here and can cause the MultiSegmentDataEntry control to parse incorrectly. By adding the space character to **IgnoreTheseCharsAfter**, it will be skipped as the data is split. In fact, the space character is a very common entry for this property.

  This property permits a list of characters. For example, if you want to ignore dash, exclamation point, and slash, just enter `"-!/"`.

  It does a case insensitive match in case you want to include letters here.

  If you enter the text `"a-z"` (in lowercase), it is a special symbol that means consider all letters (including Unicode) to be ignored. Only use it when this segment and the next visible segment do not allow letters in their TextBoxes.

  If you enter the text `"0-9"`, it is a special symbol that means consider all digits to be ignored. Only use it when this segment and the next visible segment do not allow digits in their TextBoxes.

  It defaults to "".

- **NoTextBeforeWhenBlank** (Boolean) – When getting the value from the **MultiSegmentDataEntry.Text** property, if the TextBox is blank, do not add **FormattingTextBefore** to the result when this is `true`. When `false`, add **FormattingTextBefore**.

  Only set to `true` in these cases:

  - When it is the last segment
  - When **Required** is `false` and **TextWhenBlank** has been assigned

  Otherwise there will be no text representing this segment and it will not be parsed correctly the next time.

  A good example of this is the extension on a phone number. Its usually "x" + digits. When the digits are missing, also remove the "x". When the extension is included, **MultiSegmentDataEntry.Text** returns `(###) ###-####x####`. When the extension is omitted, **MultiSegmentDataEntry.Text** returns `(###) ###-####`.

  It defaults to `false`.

- **NoTextAfterWhenBlank** (Boolean) – When getting the value from the **MultiSegmentDataEntry.Text** property, if the TextBox is blank, do not add **FormattingTextAfter** to the result when this is `true`. When `false`, add **FormattingTextAfter**.

  Only set to `true` in these cases:

  - When it is the last segment
  - When **Required** is `false` and **TextWhenBlank** has been assigned

  Otherwise there will be no text representing this segment and it will not be parsed correctly the next time.

  It defaults to `false`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **TextWhenBlank** (string) – If the TextBox is left blank and the **Required** property is `false`, this text will be used when rejoining.

  A good example of using this property: set up the area code segment of a phone number to return " " (three spaces). Phone numbers to be entered without an area code will be formatted like this: `(  )  ###-####`.

  When splitting, if this text is found (case insensitive), the TextBox is blank.

  When joining, if the TextBox is blank, this text is inserted.

  This is not used when **Required** is `true`.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Data Entry and Validation Rule Properties

- **LettersUppercase** (Boolean) – When `true`, uppercase letters are permitted. When `false`, they are filtered out when typed or reported as invalid by the MultiSegmentDataEntryValidator. It defaults to `false`.
- **LettersLowercase** (Boolean) – When `true`, lowercase letters are permitted. When `false`, they are filtered out when typed or reported as invalid by the MultiSegmentDataEntryValidator. It defaults to `false`.
- **Digits** (Boolean) – When `true`, digit characters are permitted. When `false`, they are filtered out when typed or reported as invalid by the MultiSegmentDataEntryValidator. It defaults to `true`.
- **OtherCharacters** (string) – When assigned, each character in this property is permitted in the TextBox. When `false`, they are filtered out when typed or reported as invalid by the MultiSegmentDataEntryValidator.

  Avoid putting the same characters in here as in **TabOnTheseKeys**.

  It defaults to "".
- **Required** (Boolean) – When `true`, the TextBox requires text or the MultiSegmentDataEntryValidator will report an error. When `false`, the textbox can be blank. In that case, you should consider using **TextWhenBlank** to provide some kind of marker where the data portion of the segment goes.

  It defaults to `true`.
- **MinLength** (integer) – The minimum number of characters required in this TextBox. When less than this, MultiSegmentDataEntryValidator reports an error.

  If the **Required** property is `false` and **MinLength** > 0, when the number of characters entered is 0, there will be no error.

  This value must be less than or equal **MaxLength**.

  It defaults to 0.

  Note: There is a MaxLength property too. It is in the previous section as it is essential in splitting the text.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Appearance Properties

- **DisplayTextBefore** (string) – Text that is shown before the TextBox. HTML tags are permitted. Use it to establish text and HTML formatting that appears before the TextBox.

  If **DisplayTextBefore** and **DisplayTextAfter** are both blank, **FormattingTextBefore** is used. Often **FormattingTextBefore** lacks some HTML that provides good on-screen formatting. For example, when **FormattingTextBefore** contains a left parenthesis, when shown on the page, you feel it needs to be "` (`".

  It defaults to "".

- **DisplayTextAfter** (string) – Text that is shown after the TextBox. HTML tags are permitted. Use it to establish text and HTML formatting that appears after the TextBox.

  If **DisplayTextBefore** and **DisplayTextAfter** are both blank, **FormattingTextAfter** is used. Often **FormattingTextAfter** lacks some HTML that provides good on-screen formatting. For example, when **FormattingTextAfter** contains a right parenthesis, when shown on the page, you feel it needs to be "`) `".

  It defaults to "".

- **Width** (string) – The width of the TextBox. It is used when **AutoWidth** is `false`. Use values showing either pixels or percentages.

  When "", it is not used.

  When set, it changes **AutoWidth** to `false`.

  It defaults to "".

- **AutoWidth** (Boolean) – When `true`, set the width of the TextBox by assigning the value of the **MaxLength** property to **TextBox.Columns**. When `false`, use the **Width** property. It defaults to `true`.

- **CssClass** (string) – Style sheet class name to apply to the TextBox. (It does not affect the text contributed by **FormattingTextBefore**, **DisplayTextBefore**, **FormattingTextAfter**, or **DisplayTextAfter**.)

  The MultiSegmentDataEntry control has numerous properties for the formatting all text and inputs. This property overrides those settings on the segment's TextBox.

  It defaults to "".

  One use is to hide the TextBox borders. The **DESStyleSheet.css** file supplies the style sheet class "DESMultiSegTextBox" to establish very light borders. You can use it and edit it to match your requirements. See "Changing the Appearance with Style Sheets".

- **TextAlign** (enum PeterBlum.DES.TextAlign) – By default, text is left justified in western cultures. Often users like to right justify numeric values in textboxes. This property offers justification. It adds the attribute `style='text-align:`*`value`*`;'` to the `<input type='text'>` tag.

  Some browsers do not support the text-align style and will ignore this property.

  The enumerated type `PeterBlum.DES.TextAlign` has these values:

  - `Default` – This is the default. When set to `Default`, no style=text-align attribute is written, allowing the style sheets of the page to manage it.
  - `Left`
  - `Center`
  - `Right`
  - `Justify`

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **ToolTip** (string) – The text to show in a tooltip on the TextBox. It defaults to "".

  The MultiSegmentDataEntry control has its own **ToolTip** property. If it's assigned and this **ToolTip** property is not assigned, the segment inherits the value from the MultiSegmentDataEntry control.

  When using the "Enhanced ToolTips" feature, the browser's tooltip will be replaced by a PopupView. See the **Interactive Pages User's Guide**.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Behavior Properties

- **ID** (string) – Gets the ID of the TextBox control. It is always MultiSegmentDataEntry.ID + "_" + SegmentNumber. (SegmentNumber starts at 1.) *Note: This is a read-only property.*

  Use it to programmatically assign the ID to a Validator in the `Page_Load()` method. Alternatively, you can enter the ID into the Validator's **ControlIDToEvaluate** property by using the pattern `MultiSegmentDataEntry.ID + "_" + SegmentNumber`.

- **Visible** (Boolean) – When `true`, the segment is visible. When `false`, the segment is not used except it takes up a segment number. Segments following an invisible segment are still used.

  It defaults to `true`.

- **TabOnTheseKeys** (string) – Define a list of characters that when typed, tab to the next segment. It allows the user to hit keys that are usually the formatting between segments to jump. For example, when entering a phone number in this format, `(###) ###-####`, the `")"`, space and `"-"` characters should be added here.

  It accepts a list of characters. For example, to tab on the `")"` and space character, enter `") "`.

  When a segment requires the same number of characters as in **MaxLength** and **MultiSegmentDataEntry.TabAtMaxLength** is `true`, do not use this property because it will auto-tab when the segment reaches its limit. Instead, put the characters into the next segment's **IgnoreTheseCharsBefore** property.

  Tabbing never occurs when the textbox is empty (in case the user hits several of these characters back-to-back that might advance on each TextBox).

  This property doesn't apply to the last segment.

  This is one of several tabbing features. The user can hit TAB, use **MultiSegmentDataEntry.TabAtMaxLength** and **MultiSegmentDataEntry.TabOnEnterKey**.

  It defaults to "".

- **DisableAutoComplete** (Boolean) – Several browsers provide an "autocomplete" or "autofill" feature, where a list of previous entries appears as the user starts typing. This behavior often is inappropriate as the browser is guessing to what the items are and its guess may be incorrect. For example, an integer textbox may still popup a list containing alphabetic entries. These browsers offer the ability to disable autocomplete on a field-by-field basis. Set this to `true` to disable it on this control. It defaults to `false`.

- **DisablePaste** (Boolean) – When you prefer that the user cannot paste anything into the TextBox, set this to `true`. It is supported on Internet Explorer and any other browser that supports the 'onpaste' event. It defaults to `false`.

- **TextBox** (PeterBlum.DES.FilteredTextBox) – A reference to the FilteredTextBox Control that supports the TextBox in this segment. Use it to programmatically modify any other properties not made directly available in the `PeterBlum.DES.TextSegment` class. Usually you modify its properties in the `Page_Load()` and post back event handler methods. *Note: This is a read-only property.*

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Hint Properties

License Note: This feature requires a license for the Peter's Interactive Pages.

The MultiSegmentDataEntry control supports the Interactive Hint feature. Most of its properties are defined on the control itself. See "Hint Properties" on the MultiSegmentDataEntry Control Properties. The segment defines the text of the hint.

The Properties Editor shows these properties in the "Hint" category.

- **Hint** (string) – When using the Interactive Hints system, this is the text of the hint.

  When blank, if the segment is using its **ToolTip** property, the **ToolTip** is used as the text of the hint.

  HTML tags are permitted. ENTER and LINEFEED characters are not. Use the token "{NEWLINE}" where you need a linefeed.

  When the hint is shown in the browser's status bar, HTML tags will automatically be stripped.

  It defaults to "".

- **HintLookupID** (string) – Gets the value for **Hint** through the String Lookup System. (See "The String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Hint. If no match is found OR this is blank, **Hint** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

- **HintHelp** (string) – When the Hint uses a PopupView, this provides data for use by the Help Button and other features on the PopupView. Its use depends on the **PopupView.HelpBehavior** property. (The PopupView is determined by the HintFormatter with its **PopupViewName** property.)

  The PopupView has an optional Help button. When setup, the user can click it to bring up additional information, such as a new page of help text.

  Here is how to use the **HintHelp** based on **PopupView.HelpBehavior**:

  - `None` - Do not show a Help Button. The **HintHelp** property is not used.
  - `ButtonAppends` - Add the text from **HintHelp** after the existing message. Use **PopupView.AppendHelpSeparator** to separate the two parts. When clicked, the Help button disappears and the message box is redrawn.
  - `ButtonReplaces` - Replace the text in the message with the **HintHelp**. When clicked, the Help button disappears and the message box is redrawn.
  - `Title` - The text appears in the header as the title. It replaces the **PopupView.HeaderText**. There is no Help Button. If **HintHelp** is blank, **PopupView.HeaderText** is used.
  - `Hyperlink` - Provide a Hyperlink. The Help Info text will appear in the "{0}" token of **PopupView.HyperlinkUrlForHelpButton**.

    For example, the **HyperlinkUrlForHelpButton** property may be "{0}" and this property is the complete URL "/helpfiles/helptopic1000.aspx".

    Another example uses the token for just a querystring parameter, like this: **HyperlinkUrlForHelpButton** = "/gethelp.aspx?topicid={0}" and this property contains the number of the ID.
  - `HyperlinkNewWindow` - Provide a Hyperlink that opens a new window. The **HintHelp** text will appear in the "{0}" token of **PopupView.HyperlinkUrlForHelpButton**.
  - `ButtonRunsScript` - Runs the script supplied in **PopupView.ScriptForHelpButton**. The **HintHelp** text will replace the token "{0}" in that script.

  This defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **HintHelpLookupID** (string) – Gets the value for **HintHelp** through the String Lookup System. (See "The String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Hint. If no match is found OR this is blank, **HintHelp** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties for the PeterBlum.DES.IntegerTextSegment Class

For directions on creating a `PeterBlum.DES.IntegerTextSegment` object, see step 0 in "Adding a MultiSegmentDataEntry Control".

By default, no properties are saved in the control's ViewState. If you set them programmatically, you must do it every time. Also you can save an individual property in the ViewState by calling the `TrackPropertyInViewState("propertyname")` method on the Segment object.

Click on any of these topics to jump to them:

- Get and Set Text Properties
- Split and Join Text Properties
- Data Entry and Validation Rules Properties
- Appearance Properties
- Behavior Properties
- Spinners Properties
- Hint Properties

Get and Set Text Properties

These three properties interact directly with the TextBox's text value. Usually you will assign a string to the **MultiSegmentDataEntry.Text** and **TextNoSeparator** properties and they will split and join the string amongst the segments. See the next topic. Use these when you are working with something other than a string whose elements you convert to text and are associated with individual segments.

- **Text** (string) – Gets and sets the value assigned to the TextBox. It does not validate or parse the text. If you want the MultiSegmentDataEntry control to parse a string, assign your string to **MultiSegmentDataEntry.Text** or **MultiSegmentDataEntry.TextNoSeparator**.

  Use this when MultiSegmentDataEntry does not handle your data format. For example, when editing a date, you may start with a `System.DateTime` structure, not a string. So convert each element – day, month, and year – to a string, then assign it here.

- **IsValid** (Boolean) – Returns `true` when the TextBox matches the segment's validation rules, like **Required**, the character set, and text length. Returns `false`, when it fails to match the validation rules. Usually you will attach a MultiSegmentDataEntryValidator to the MultiSegmentDataEntry control to validate it.

- **IsEmpty** (Boolean) – Returns `true` when the TextBox is blank. Returns `false` when it has text. When the TextBox contains a value matching **TextWhenBlank**, it is not considered empty.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Split and Join Text Properties

These properties support the **MultiSegmentDataEntry.Text** and **TextNoSeparator** properties as they split and join one string amongst the segments.

- **MaxLength** (integer) – The maximum number of characters required in this segment. A maximum is always required. It provides a limit for splitting the text when you set the **MultiSegmentDataEntry.Text** property.

  This value will be assigned to the TextBox's **MaxLength** property imposing a client-side limit as the user types.

  It defaults to 4.

  Note: There is a MinLength property too. It is not used to split and join but it is used for validation. See the text topic.

- **FormattingTextBefore** (string) – Text that appears before the value that goes into the TextBox. It is used to split and join the string you supply in the **MultiSegmentDataEntry.Text** property.

  When splitting text, if this string is found, it is skipped so the text after it will be assigned to the TextBox.

  When joining text, **FormattingTextBefore** precedes the value from the TextBox.

  If **DisplayTextBefore** is blank, this property is shown on the page before the TextBox.

  It defaults to "".

- **FormattingTextAfter** (string) – Text that appears after the value that goes into the TextBox. It is used to split and join the string you supply in the **MultiSegmentDataEntry.Text** property.

  When splitting text, if this string is found, it is skipped so the text after it will be assigned to the next segment.

  When joining text, **FormattingTextAfter** follows the value from the TextBox.

  If **DisplayTextAfter** is blank, this property is shown on the page after the TextBox.

  It defaults to "".

- **IgnoreTheseCharsBefore** (Boolean) – When setting the **MultiSegmentDataEntry.Text** property, the control splits your string amongst each segment. Each character in **IgnoreTheseCharsBefore** will be ignored if found preceding the value that is assigned to the TextBox. It helps the MultiSegmentDataEntry control support badly formatted data.

  When splitting, data you retrieve from your database may contain inappropriate characters, due to inconsistent data entry rules. Perhaps another web page or application was used to collect this data and its validation rules did not stop entries that you now consider illegal formats.

  For example, you demand phone numbers to have this format: `(###) ###-####`. But your database contains phone numbers with additional spaces: `( ### ) ### - ####`. The space character is inappropriate here and can cause the MultiSegmentDataEntry control to parse incorrectly. By adding the space character to **IgnoreTheseCharsBefore**, it will be skipped as the data is split. In fact, the space character is a very common entry for this property.

  This property permits a list of characters. For example, if you want to ignore dash, exclamation point, and slash, just enter `"-!/"`.

  It does a case insensitive match in case you want to include letters here.

  If you enter the text `"a-z"` (in lowercase), it is a special symbol that means consider all letters (including Unicode) to be ignored. Only use it when this segment and the next visible segment do not allow letters in their TextBoxes.

  If you enter the text `"0-9"`, it is a special symbol that means consider all digits to be ignored. Only use it when this segment and the next visible segment do not allow digits in their TextBoxes.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **IgnoreTheseCharsAfter** (Boolean) – When setting the **MultiSegmentDataEntry.Text** property, the control splits your string amongst each segment. Each character in **IgnoreTheseCharsAfter** will be ignored if found after the value that is assigned to the TextBox. It helps the MultiSegmentDataEntry control support badly formatted data.

  When splitting, data you retrieve from your database may contain inappropriate characters, due to inconsistent data entry rules. Perhaps another web page or application was used to collect this data and its validation rules did not stop entries that you now consider illegal formats.

  For example, you demand phone numbers to have this format: `(###) ###-####`. But your database contains phone numbers with additional spaces: `( ### ) ### - ####`. The space character is inappropriate here and can cause the MultiSegmentDataEntry control to parse incorrectly. By adding the space character to **IgnoreTheseCharsAfter**, it will be skipped as the data is split. In fact, the space character is a very common entry for this property.

  This property permits a list of characters. For example, if you want to ignore dash, exclamation point, and slash, just enter `"-!/"`.

  It does a case insensitive match in case you want to include letters here.

  If you enter the text `"a-z"` (in lowercase), it is a special symbol that means consider all letters (including Unicode) to be ignored. Only use it when this segment and the next visible segment do not allow letters in their TextBoxes.

  If you enter the text `"0-9"`, it is a special symbol that means consider all digits to be ignored. Only use it when this segment and the next visible segment do not allow digits in their TextBoxes.

  It defaults to "".

- **NoTextBeforeWhenBlank** (Boolean) – When getting the value from the **MultiSegmentDataEntry.Text** property, if the TextBox is blank, do not add **FormattingTextBefore** to the result when this is `true`. When `false`, add **FormattingTextBefore**.

  Only set to `true` in these cases:

  - When it is the last segment
  - When **Required** is `false` and **TextWhenBlank** has been assigned

  Otherwise there will be no text representing this segment and it will not be parsed correctly the next time.

  A good example of this is the extension on a phone number. Its usually "x" + digits. When the digits are missing, also remove the "x". When the extension is included, **MultiSegmentDataEntry.Text** returns `(###) ###-####x####`. When the extension is omitted, **MultiSegmentDataEntry.Text** returns `(###) ###-####`.

  It defaults to `false`.

- **NoTextAfterWhenBlank** (Boolean) – When getting the value from the **MultiSegmentDataEntry.Text** property, if the TextBox is blank, do not add **FormattingTextAfter** to the result when this is `true`. When `false`, add **FormattingTextAfter**.

  Only set to `true` in these cases:

  - When it is the last segment
  - When **Required** is `false` and **TextWhenBlank** has been assigned

  Otherwise there will be no text representing this segment and it will not be parsed correctly the next time.

  It defaults to `false`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **TextWhenBlank** (string) – If the TextBox is left blank and the **Required** property is `false`, this text will be used when rejoining.

  A good example of using this property: set up the area code segment of a phone number to return " " (three spaces). Phone numbers to be entered without an area code will be formatted like this: `(  )  ###-####`.

  When splitting, if this text is found (case insensitive), the TextBox is blank.

  When joining, if the TextBox is blank, this text is inserted.

  This is not used when **Required** is `true`.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Data Entry and Validation Rules Properties

- **Required** (Boolean) – When `true`, the TextBox requires text or the MultiSegmentDataEntryValidator will report an error. When `false`, the textbox can be blank. In that case, you should consider using **TextWhenBlank** to provide some kind of marker where the data portion of the segment goes.

  It defaults to `true`.

- **AllowNegatives** (Boolean) – Determines if negative numbers are permitted. When `true`, they are permitted. It defaults to `true`. When `false`, keyboard filtering will not allow the minus ("-") character, and the MultiSegmentDataEntryValidator will report errors when negative values are entered. It defaults to `false`.

- **FillLeadZeros** (Integer) – Provides additional formatting when converting an integer to text by adding lead zeros. When > 0, it adds enough lead zeroes to match the value of this property. For example, if this is 4, all values will have 4 digits. Any number that does not offer 4 digits gets lead zeros to fill it. When the user enters "34", this reformats to "0034".

  The lead zeros are preserved when you **MultiSegmentDataEntry.Text** splits and joins.

  When 0, it is not used.

  It defaults to 0.

- **MinLength** (integer) – The minimum number of characters required in this TextBox. When less than this, MultiSegmentDataEntryValidator reports an error.

  If the **Required** property is `false` and **MinLength** > 0, when the number of characters entered is 0, there will be no error.

  This value must be less than or equal **MaxLength**.

  It defaults to 0.

  Note: There is a MaxLength property too. It is in the previous section as it is essential in splitting the text.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Appearance Properties

- **DisplayTextBefore** (string) – Text that is shown before the TextBox. HTML tags are permitted. Use it to establish text and HTML formatting that appears before the TextBox.

  If blank, **FormattingTextBefore** is used. Often **FormattingTextBefore** lacks some HTML that provides good onscreen formatting. For example, when **FormattingTextBefore** contains a left parenthesis, when shown on the page, you feel it needs to be "` (`".

  It defaults to "".

- **DisplayTextAfter** (string) – Text that is shown after the TextBox. HTML tags are permitted. Use it to establish text and HTML formatting that appears after the TextBox.

  If blank, **FormattingTextAfter** is used. Often **FormattingTextAfter** lacks some HTML that provides good onscreen formatting. For example, when **FormattingTextAfter** contains a right parenthesis, when shown on the page, you feel it needs to be "`) `".

  It defaults to "".

- **Width** (string) – The width of the TextBox. It is used when **AutoWidth** is `false`. Use values showing either pixels or percentages.

  When "", it is not used.

  When set, it changes **AutoWidth** to `false`.

  It defaults to "".

- **AutoWidth** (Boolean) – When `true`, set the width of the TextBox by assigning the value of the **MaxLength** property to **TextBox.Columns**. When `false`, use the **Width** property. It defaults to `true`.

- **CssClass** (string) – Style sheet class name to apply to the TextBox. (It does not affect the text contributed by **FormattingTextBefore**, **DisplayTextBefore**, **FormattingTextAfter**, or **DisplayTextAfter**.)

  The MultiSegmentDataEntry control has numerous properties for the formatting all text and inputs. This property overrides those settings on the segment's TextBox.

  It defaults to "".

  One use is to hide the TextBox borders. The **DESStyleSheet.css** file supplies the style sheet class "DESMultiSegTextBox" to establish very light borders. You can use it and edit it to match your requirements. See "Changing the Appearance with Style Sheets".

  Note: DES 2.0 users must add "DESMultiSegTextBox" to their style sheet file.

- **TextAlign** (enum PeterBlum.DES.TextAlign) – By default, text is left justified in western cultures. Often users like to right justify numeric values in textboxes. This property offers justification. It adds the attribute `style='text-align:`*`value`*`;'` to the `<input type='text'>` tag.

  Some browsers do not support the text-align style and will ignore this property.

  The enumerated type `PeterBlum.DES.TextAlign` has these values:

  - `Default` – This is the default. When set to `Default`, no style=text-align attribute is written, allowing the style sheets of the page to manage it.
  - `Left`
  - `Center`
  - `Right`
  - `Justify`

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **ToolTip** (string) – The text to show in a tooltip on the TextBox. It defaults to "".

  The MultiSegmentDataEntry control has its own **ToolTip** property. If it's assigned and this **ToolTip** property is not assigned, the segment inherits the value from the MultiSegmentDataEntry control.

  When using the "Enhanced ToolTips" feature, the browser's tooltip will be replaced by a PopupView. See the **Interactive Pages User's Guide**.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Behavior Properties

- **ID** (string) – Gets the ID of the TextBox control. It is always MultiSegmentDataEntry.ID + "_" + SegmentNumber. (SegmentNumber starts at 1.) *Note: This is a read-only property.*

  Use it to programmatically assign the ID to a Validator. Alternatively, you can enter the ID into the Validator's **ControlIDToEvaluate** property by using the pattern `MultiSegmentDataEntry.ID + "_" + SegmentNumber`.

- **Visible** (Boolean) – When `true`, the segment is visible. When `false`, the segment is not used except it takes up a segment number. Segments following an invisible segment are still used.

  It defaults to `true`.

- **TabOnTheseKeys** (string) – Define a list of characters that when typed, tab to the next segment. It allows the user to hit keys that are usually the formatting between segments to jump. For example, when entering a phone number in this format, `(###) ###-####`, the "`)`", space and "`-`" characters should be added here.

  It accepts a list of characters. For example, to tab on the "`)`" and space character, enter "`) `".

  When a segment requires the same number of characters as in **MaxLength** and **MultiSegmentDataEntry.TabAtMaxLength** is `true`, do not use this property because it will auto-tab when the segment reaches its limit. Instead, put the characters into the next segment's **IgnoreTheseCharsBefore** property.

  Tabbing never occurs when the textbox is empty (in case the user hits several of these characters back-to-back that might advance on each TextBox).

  This property doesn't apply to the last segment.

  This is one of several tabbing features. The user can hit TAB, use **MultiSegmentDataEntry.TabAtMaxLength** and **MultiSegmentDataEntry.TabOnEnterKey**.

  It defaults to "".

- **DisableAutoComplete** (Boolean) – Several browsers provide an "autocomplete" or "autofill" feature, where a list of previous entries appears as the user starts typing. This behavior often is inappropriate as the browser is guessing to what the items are and its guess may be incorrect. For example, an integer textbox may still popup a list containing alphabetic entries. These browsers offer the ability to disable autocomplete on a field-by-field basis. Set this to `true` to disable it on this control. It defaults to `false`.

- **DisablePaste** (Boolean) – When you prefer that the user cannot paste anything into the TextBox, set this to `true`. It is supported on Internet Explorer and any other browser that supports the 'onpaste' event. It defaults to `false`.

- **TextBox** (PeterBlum.DES.IntegerTextBox) – A reference to the IntegerTextBox control that supports the TextBox in this segment. Use it to programmatically modify any other properties not made directly available in the `PeterBlum.DES.IntegerTextSegment` class. Usually you modify its properties in the `Page_Load()` and post back event handler methods. *Note: This is a read-only property.*

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Spinners Properties

The Spinner is an extension to the `PeterBlum.DES.IntegerTextBox`. It provides a pair of arrow buttons that increment or decrement the value of the TextBox when clicked.

The Spinner is an extension to the IntegerSegment. It provides a pair of arrow buttons that increment or decrement the value of the textbox when clicked. You can customize the button appearance and autorepeat speed with properties on **PeterBlum.DES.Globals.Page.SpinnerManager**.

- **ShowSpinner** (Boolean) – When `true`, the spinner control is shown. When `false`, it is not. It defaults to `false`.

  Note: Spinners are only supported on these browsers: IE Windows 5+, Netscape 7+, Mozilla 1.1+, FireFox, Opera 7+, and Safari. Other browsers will ignore this property.

- **IncrementValue** (Double) – The number to add or subtract to the current value. It supports decimal values. It defaults to 1.0.

- **SpinnerMinValue** (String) - Establishes a minimum value for the spinner. When it reaches this number, it stops spinning. Leave blank when not used. Otherwise enter a decimal value.

  When the textbox is blank and the up arrow is hit, this is the first value shown in the textbox.

  It defaults to "".

- **SpinnerMaxValue** (String) - Establishes a maximum value for the spinner. When it reaches this number, it stops spinning. Leave blank when not used. Otherwise enter a decimal value.

  When the textbox is blank and the down arrow is hit, this is the first value shown in the textbox.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Hint Properties

License Note: This feature requires a license for the Peter's Interactive Pages.

The MultiSegmentDataEntry control supports the Interactive Hint feature. Most of its properties are defined on the control itself. See "Hint Properties" on the MultiSegmentDataEntry Control Properties. The segment defines the text of the hint.

The Properties Editor shows these properties in the "Hint" category.

- **Hint** (string) – When using the Interactive Hints system, this is the text of the hint.

  When blank, if the segment is using its **ToolTip** property, the **ToolTip** is used as the text of the hint.

  HTML tags are permitted. ENTER and LINEFEED characters are not. When the hint is shown in the browser's status bar, HTML tags will automatically be stripped.

  It defaults to "".

- **HintLookupID** (string) – Gets the value for **Hint** through the String Lookup System. (See "The String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Hint. If no match is found OR this is blank, **Hint** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

- **HintHelp** (string) – When the Hint uses a PopupView, this provides data for use by the Help Button and other features on the PopupView. Its use depends on the **PopupView.HelpBehavior** property. (The PopupView is determined by the HintFormatter with its **PopupViewName** property.)

  The PopupView has an optional Help button. When setup, the user can click it to bring up additional information, such as a new page of help text.

  Here is how to use the **HintHelp** based on **PopupView.HelpBehavior**:

  - `None` - Do not show a Help Button. The **HintHelp** property is not used.
  - `ButtonAppends` - Add the text from **HintHelp** after the existing message. Use **PopupView.AppendHelpSeparator** to separate the two parts. When clicked, the Help button disappears and the message box is redrawn.
  - `ButtonReplaces` - Replace the text in the message with the **HintHelp**. When clicked, the Help button disappears and the message box is redrawn.
  - `Title` - The text appears in the header as the title. It replaces the **PopupView.HeaderText**. There is no Help Button. If **HintHelp** is blank, **PopupView.HeaderText** is used.
  - `Hyperlink` - Provide a Hyperlink. The Help Info text will appear in the "{0}" token of **PopupView.HyperlinkUrlForHelpButton**.

    For example, the **HyperlinkUrlForHelpButton** property may be "{0}" and this property is the complete URL "/helpfiles/helptopic1000.aspx".

    Another example uses the token for just a querystring parameter, like this: **HyperlinkUrlForHelpButton** = "/gethelp.aspx?topicid={0}" and this property contains the number of the ID.
  - `HyperlinkNewWindow` - Provide a Hyperlink that opens a new window. The **HintHelp** text will appear in the "{0}" token of **PopupView.HyperlinkUrlForHelpButton**.
  - `ButtonRunsScript` - Runs the script supplied in **PopupView.ScriptForHelpButton**. The **HintHelp** text will replace the token "{0}" in that script.

  This defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **HintHelpLookupID** (string) – Gets the value for **HintHelp** through the String Lookup System. (See "The String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Hint. If no match is found OR this is blank, **HintHelp** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties for the PeterBlum.DES.DropDownListSegment Class

For directions on creating a `PeterBlum.DES.DropDownListSegment` object, see step 0 in "Adding a MultiSegmentDataEntry Control".

Remember that DropDownLists have two values associated with each item: the text shown to the user and the value. The MultiSegmentDataEntry control uses the value, not the text shown to the user.

By default, only the **Items** property is saved in the control's ViewState. If you set other properties programmatically, you must do it every time. Also you can save an individual property in the ViewState by calling the `TrackPropertyInViewState("`*`propertyname`*`")` method on the Segment object.

Click on any of these topics to jump to them:

- Get and Set Text Properties
- Split and Join Text Properties
- DropDownList Items Properties
- Data Entry and Validation Rules Properties
- Appearance Properties
- Behavior Properties
- Hint Properties

Get and Set Text Properties

These three properties interact directly with the DropDownList's current value. Usually you will assign a string to the **MultiSegmentDataEntry.Text** and **TextNoSeparator** properties and they will split and join the string amongst the segments. See the next topic. Use these when you are working with something other than a string whose elements you convert to text and are associated with individual segments.

- **Text** (string) – Gets and sets the value assigned to the DropDownList. It does not validate or parse the text. If you want the MultiSegmentDataEntry control to parse a string, assign your string to **MultiSegmentDataEntry.Text** or **MultiSegmentDataEntry.TextNoSeparator**.

  When getting text, if there is no selection, it returns "".

  Use this when MultiSegmentDataEntry does not handle your data format. For example, when editing a date, you may start with a `System.DateTime` structure, not a string. So convert each element – day, month, and year – to a string, then assign it here.

- **IsValid** (Boolean) – Returns `true` when the DropDownList matches the segment's validation rules, like **Required**, the character set, and text length. Returns `false`, when it fails to match the validation rules. Usually you will attach a MultiSegmentDataEntryValidator to the MultiSegmentDataEntry control to validate it.

- **IsEmpty** (Boolean) – Returns `true` when the DropDownList has no selection. Returns `false` when it has a selection. When the DropDownList contains a value matching **TextWhenBlank**, it is not considered empty.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Split and Join Text Properties

These properties support the **MultiSegmentDataEntry.Text** and **TextNoSeparator** properties as they split and join one string amongst the segments.

- **MaxLength** (integer) – The maximum number of characters required in this segment. A maximum is always required. It provides a limit for splitting the text when you set the **MultiSegmentDataEntry.Text** property.

  It defaults to 4.

- **FormattingTextBefore** (string) – Text that appears before the value that goes into the DropDownList. It is used to split and join the string you supply in the **MultiSegmentDataEntry.Text** property.

  When splitting text, if this string is found, it is skipped so the text after it will be assigned to the DropDownList.

  When joining text, **FormattingTextBefore** precedes the value from the DropDownList.

  If **DisplayTextBefore** is blank, this property is shown on the page before the DropDownList.

  It defaults to "".

- **FormattingTextAfter** (string) – Text that appears after the value that goes into the DropDownList. It is used to split and join the string you supply in the **MultiSegmentDataEntry.Text** property.

  When splitting text, if this string is found, it is skipped so the text after it will be assigned to the next segment.

  When joining text, **FormattingTextAfter** follows the value from the DropDownList.

  If **DisplayTextAfter** is blank, this property is shown on the page after the DropDownList.

  It defaults to "".

- **IgnoreTheseCharsBefore** (Boolean) – When setting the **MultiSegmentDataEntry.Text** property, the control splits your string amongst each segment. Each character in **IgnoreTheseCharsBefore** will be ignored if found preceding the value that is assigned to the DropDownList. It helps the MultiSegmentDataEntry control support badly formatted data.

  When splitting, data you retrieve from your database may contain inappropriate characters, due to inconsistent data entry rules. Perhaps another web page or application was used to collect this data and its validation rules did not stop entries that you now consider illegal formats.

  For example, you demand phone numbers to have this format: `(###) ###-####`. But your database contains phone numbers with additional spaces: `( ### ) ### - ####`. The space character is inappropriate here and can cause the MultiSegmentDataEntry control to parse incorrectly. By adding the space character to **IgnoreTheseCharsBefore**, it will be skipped as the data is split. In fact, the space character is a very common entry for this property.

  This property permits a list of characters. For example, if you want to ignore dash, exclamation point, and slash, just enter `"-!/"`.

  It does a case insensitive match in case you want to include letters here.

  If you enter the text `"a-z"` (in lowercase), it is a special symbol that means consider all letters (including Unicode) to be ignored. Only use it when this segment and the next visible segment do not allow letters in their DropDownLists.

  If you enter the text `"0-9"`, it is a special symbol that means consider all digits to be ignored. Only use it when this segment and the next visible segment do not allow digits in their DropDownLists.

  It defaults to "".

- **IgnoreTheseCharsAfter** (Boolean) – When setting the **MultiSegmentDataEntry.Text** property, the control splits your string amongst each segment. Each character in **IgnoreTheseCharsAfter** will be ignored if found after the value that is assigned to the DropDownList. It helps the MultiSegmentDataEntry control support badly formatted data.

  When splitting, data you retrieve from your database may contain inappropriate characters, due to inconsistent data entry rules. Perhaps another web page or application was used to collect this data and its validation rules did not stop entries that you now consider illegal formats.

  For example, you demand phone numbers to have this format: `(###) ###-####`. But your database contains phone numbers with additional spaces: `( ### ) ### - ####`. The space character is inappropriate here and can cause the

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

MultiSegmentDataEntry control to parse incorrectly. By adding the space character to **IgnoreTheseCharsAfter**, it will be skipped as the data is split. In fact, the space character is a very common entry for this property.

This property permits a list of characters. For example, if you want to ignore dash, exclamation point, and slash, just enter `"-!/"`.

It does a case insensitive match in case you want to include letters here.

If you enter the text `"a-z"` (in lowercase), it is a special symbol that means consider all letters (including Unicode) to be ignored. Only use it when this segment and the next visible segment do not allow letters in their DropDownLists.

If you enter the text `"0-9"`, it is a special symbol that means consider all digits to be ignored. Only use it when this segment and the next visible segment do not allow digits in their DropDownLists.

It defaults to "".

- **NoTextBeforeWhenBlank** (Boolean) – When getting the value from the **MultiSegmentDataEntry.Text** property, if the DropDownList is blank, do not add **FormattingTextBefore** to the result when this is `true`. When `false`, add **FormattingTextBefore**.

  Only set to `true` in these cases:

  - When it's the last segment
  - When **Required** is `false` and **TextWhenBlank** has been assigned

  Otherwise there will be no text representing this segment and it will not be parsed correctly the next time.

  A good example of this is the extension on a phone number. Its usually "x" + digits. When the digits are missing, also remove the "x". When the extension is included, **MultiSegmentDataEntry.Text** returns `(###) ###-####x####`. When the extension is omitted, **MultiSegmentDataEntry.Text** returns `(###) ###-####`.

  It defaults to `false`.

- **NoTextAfterWhenBlank** (Boolean) – When getting the value from the **MultiSegmentDataEntry.Text** property, if the DropDownList is blank, do not add **FormattingTextAfter** to the result when this is `true`. When `false`, add **FormattingTextAfter**.

  Only set to `true` in these cases:

  - When it's the last segment
  - When **Required** is `false` and **TextWhenBlank** has been assigned

  Otherwise there will be no text representing this segment and it will not be parsed correctly the next time.

  It defaults to `false`.

- **TextWhenBlank** (string) – If the DropDownList has no selection and the **Required** property is `false`, this text will be used when rejoining.

  A good example of using this property: set up the area code segment of a phone number to return " " (three spaces). Phone numbers to be entered without an area code will be formatted like this: `(  ) ###-####`.

  When splitting, if this text is found (case insensitive), the DropDownList is blank.

  When joining, if the DropDownList is blank, this text is inserted.

  This is not used when **Required** is `true`.

  It defaults to "".

- **LettersUppercase** (Boolean) – Used when splitting text. When `true`, uppercase letters are permitted. When `false` and found in the text, the end of this segment's text has been reached. It defaults to `false`.

- **LettersLowercase** (Boolean) – Used when splitting text. When `true`, lowercase letters are permitted. When `false` and found in the text, the end of this segment's text has been reached. It defaults to `false`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **Digits** (Boolean) – Used when splitting text. When `true`, digit characters are permitted. When `false` and found in the text, the end of this segment's text has been reached. It defaults to `true`.
- **OtherCharacters** (string) – Used assigned, each character in this property is permitted in the DropDownList value. When "" and found in the text, the end of this segment's text has been reached.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

DropDownList Items Properties

The DropDownList control within this segment is the `System.Web.UI.WebControls.DropDownList` class. The properties that set up the items shown in the list are offered here: **Items**, **DataSource**, **DataMember**, **DataTextField**, **DataValueField**, and **DataTextFormatString**. See System.Web.UI.WebControls.DropDownLists Members for details.

When you use the **DataSource** property, you must call the `MultiSegmentDataEntry.DataBind()` method to convert its data into the **Items** collection. Do this in the `Page_Load()` method when **Page.IsPostBack** is `false` or post back event handler method when the list changes after post back. The ViewState will preserve the **Items** collection between post backs.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Data Entry and Validation Rules Properties

- **Required** (Boolean) – When `true`, the DropDownList requires a selection or the MultiSegmentDataEntryValidator will report an error. When `false`, the DropDownList can have no selection. In that case, you should consider using **TextWhenBlank** to provide some kind of marker where the data portion of the segment goes.

  It defaults to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Appearance Properties

- **DisplayTextBefore** (string) – Text that is shown before the DropDownList. HTML tags are permitted. Use it to establish text and HTML formatting that appears before the DropDownList.

  If blank, **FormattingTextBefore** is used. Often **FormattingTextBefore** lacks some HTML that provides good onscreen formatting. For example, when **FormattingTextBefore** contains a left parenthesis, when shown on the page, you feel it needs to be "` (`".

  It defaults to "".

- **DisplayTextAfter** (string) – Text that is shown after the DropDownList. HTML tags are permitted. Use it to establish text and HTML formatting that appears after the DropDownList.

  If blank, **FormattingTextAfter** is used. Often **FormattingTextAfter** lacks some HTML that provides good onscreen formatting. For example, when **FormattingTextAfter** contains a right parenthesis, when shown on the page, you feel it needs to be "`) `".

  It defaults to "".

- **Width** (string) – The width of the DropDownList. Use values showing either pixels or percentages.

  When "", it is not used.

  It defaults to "".

- **CssClass** (string) – Style sheet class name to apply to the DropDownList. (It does not affect the text contributed by **FormattingTextBefore**, **DisplayTextBefore**, **FormattingTextAfter**, or **DisplayTextAfter**.)

  The MultiSegmentDataEntry control has numerous properties for the formatting all text and inputs. This property overrides those settings on the segment's DropDownList.

  It defaults to "".

- **ToolTip** (string) – The text to show in a tooltip on the DropDownList. It defaults to "".

  The MultiSegmentDataEntry control has its own **ToolTip** property. If it's assigned and this **ToolTip** property is not assigned, the segment inherits the value from the MultiSegmentDataEntry control.

  When using the "Enhanced ToolTips" feature, the browser's tooltip will be replaced by a PopupView. See the **Interactive Pages User's Guide**.

  Note: Internet Explorer for Windows v5+ never shows a tooltip on DropDownLists.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Behavior Properties

- **ID** (string) – Gets the ID of the DropDownList control. It is always MultiSegmentDataEntry.ID + "_" + SegmentNumber. (SegmentNumber starts at 1.) *Note: This is a read-only property*.

  Use it to programmatically assign the ID to a Validator. Alternatively, you can enter the ID into the Validator's **ControlIDToEvaluate** property by using the pattern `MultiSegmentDataEntry.ID + "_" + SegmentNumber`.

- **Visible** (Boolean) – When `true`, the segment is visible. When `false`, the segment is not used except it takes up a segment number. Segments following an invisible segment are still used.

  It defaults to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Hint Properties

License Note: This feature requires a license for the Peter's Interactive Pages.

The MultiSegmentDataEntry control supports the Interactive Hint feature. Most of its properties are defined on the control itself. See "Hint Properties" on the MultiSegmentDataEntry Control Properties. The segment defines the text of the hint.

The Properties Editor shows these properties in the "Hint" category.

- **Hint** (string) – When using the Interactive Hints system, this is the text of the hint.

  When blank, if the segment is using its **ToolTip** property, the **ToolTip** is used as the text of the hint.

  HTML tags are permitted. ENTER and LINEFEED characters are not. When the hint is shown in the browser's status bar, HTML tags will automatically be stripped.

  It defaults to "".

- **HintLookupID** (string) – Gets the value for **Hint** through the String Lookup System. (See "The String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Hint. If no match is found OR this is blank, **Hint** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

- **HintHelp** (string) – When the Hint uses a PopupView, this provides data for use by the Help Button and other features on the PopupView. Its use depends on the **PopupView.HelpBehavior** property. (The PopupView is determined by the HintFormatter with its **PopupViewName** property.)

  The PopupView has an optional Help button. When setup, the user can click it to bring up additional information, such as a new page of help text.

  Here is how to use the **HintHelp** based on **PopupView.HelpBehavior**:

  - `None` - Do not show a Help Button. The **HintHelp** property is not used.
  - `ButtonAppends` - Add the text from **HintHelp** after the existing message. Use **PopupView.AppendHelpSeparator** to separate the two parts. When clicked, the Help button disappears and the message box is redrawn.
  - `ButtonReplaces` - Replace the text in the message with the **HintHelp**. When clicked, the Help button disappears and the message box is redrawn.
  - `Title` - The text appears in the header as the title. It replaces the **PopupView.HeaderText**. There is no Help Button. If **HintHelp** is blank, **PopupView.HeaderText** is used.
  - `Hyperlink` - Provide a Hyperlink. The Help Info text will appear in the "{0}" token of **PopupView.HyperlinkUrlForHelpButton**.

    For example, the **HyperlinkUrlForHelpButton** property may be "{0}" and this property is the complete URL "/helpfiles/helptopic1000.aspx".

    Another example uses the token for just a querystring parameter, like this: **HyperlinkUrlForHelpButton** = "/gethelp.aspx?topicid={0}" and this property contains the number of the ID.
  - `HyperlinkNewWindow` - Provide a Hyperlink that opens a new window. The **HintHelp** text will appear in the "{0}" token of **PopupView.HyperlinkUrlForHelpButton**.
  - `ButtonRunsScript` - Runs the script supplied in **PopupView.ScriptForHelpButton**. The **HintHelp** text will replace the token "{0}" in that script.

  This defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **HintHelpLookupID** (string) – Gets the value for **HintHelp** through the String Lookup System. (See "The String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Hint. If no match is found OR this is blank, **HintHelp** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Examples

Click on any of these topics to jump to them:

- U.S./Canada Phone Number
- IP Address
- Date

U.S./Canada Phone Number

This example includes the phone number extension in the last segment.

Here are some of the rules it includes:

- There are 4 segments. Each uses a TextBox that is limited to digits. Segment 1 and 2 require 3 digits. Segment 3 requires 4 digits. Segment 4 is optional and allows up to 5 digits.
- Segment 1, the area code, has parenthesis surrounding it. It provides several other characters for the **IgnoreTheseCharsBefore**/**After** properties. It tabs on certain characters.
- Segment 2, the local exchange, has a space before it and dash after. It provides several other characters for the **IgnoreTheseCharsBefore**/**After** properties. It tabs on certain characters.
- Segment 3 tabs on certain characters
- Segment 4 shows the text “ext:” for the extension but uses the text “x” for **FormattedTextBefore**.
- Style sheets have been applied to establish a border around the entire control and dim the borders on each textbox.

```
<des:MultiSegmentDataEntry id="MultiSegmentDataEntry1" runat="server"
  CssClass="DESTBMultiSegContainer">
<Segments>
  <des:TextSegment CssClass="DESTBMultiSegTextBox"
     IgnoreTheseCharsBefore=" " IgnoreTheseCharsAfter=" "
     FormattingTextBefore="(" FormattingTextAfter=") "
     DisplayTextBefore="(" DisplayTextAfter=")"
     AutoWidth="False" Width="30px" MinLength="3" MaxLength="3"
     TabOnTheseKeys=") -" ></des:TextSegment>
  <des:TextSegment CssClass="DESTBMultiSegTextBox"
     IgnoreTheseCharsAfter=" " FormattingTextAfter="-"
     AutoWidth="False" Width="30px" MinLength="3" MaxLength="3"
     TabOnTheseKeys="- " ></des:TextSegment>
  <des:TextSegment CssClass="DESTBMultiSegTextBox"
     IgnoreTheseCharsAfter=" "
     AutoWidth="False" Width="40px" MinLength="4" MaxLength="4"
     TabOnTheseKeys="ext:" ></des:TextSegment>
  <des:TextSegment CssClass="DESTBMultiSegTextBox"
     FormattingTextBefore=" x" DisplayTextBefore="&amp;nbsp;ext:"
     MaxLength="5" Required="False"
     NoTextBeforeWhenBlank="True" ></des:TextSegment>
</Segments>
</des:MultiSegmentDataEntry>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

IP Address

A four segment field where all allow only integers. Ranges are established both on the optional spinners and the RangeValidators attached to individual segments. (You could use a MultiConditionValidator to provide a single error message. It would use 4 RangeConditions to evaluate each textbox.)

192 . 168 . 0 . 100

```
<des:MultiSegmentDataEntry id="MultiSegmentDataEntry1" runat="server">
<Segments>
  <des:IntegerTextSegment FormattingTextAfter="." TabOnTheseKeys="."
     ShowSpinner="True" SpinnerMinValue="0" SpinnerMaxValue="255"
     MaxLength="3"></des:IntegerTextSegment>
  <des:IntegerTextSegment FormattingTextAfter="." TabOnTheseKeys="."
     ShowSpinner="True" SpinnerMinValue="0" SpinnerMaxValue="255"
     MaxLength="3"></des:IntegerTextSegment>
  <des:IntegerTextSegment FormattingTextAfter="." TabOnTheseKeys="."
     ShowSpinner="True" SpinnerMinValue="0" SpinnerMaxValue="255"
     MaxLength="3"></des:IntegerTextSegment>
  <des:IntegerTextSegment ShowSpinner="True" SpinnerMinValue="0"
     SpinnerMaxValue="255" MaxLength="3"></des:IntegerTextSegment>
</Segments>
</des:MultiSegmentDataEntry>
<des:RangeValidator id="RangeValidator1" runat="server"
  ErrorMessage="Between 0 and 255"
  ControlIDToEvaluate="MultiSegmentDataEntry1_1" DataType="Integer"
  Minimum="0" Maximum="255">
</des:RangeValidator>
<des:RangeValidator id="RangeValidator2" runat="server"
  ErrorMessage="Between 0 and 255"
  ControlIDToEvaluate="MultiSegmentDataEntry1_2" DataType="Integer"
  Minimum="0" Maximum="255">
</des:RangeValidator>
<des:RangeValidator id="RangeValidator3" runat="server"
  ErrorMessage="Between 0 and 255"
  ControlIDToEvaluate="MultiSegmentDataEntry1_3" DataType="Integer"
  Minimum="0" Maximum="255">
</des:RangeValidator>
<des:RangeValidator id="RangeValidator4" runat="server"
  ErrorMessage="Between 0 and 255"
  ControlIDToEvaluate="MultiSegmentDataEntry1_4" DataType="Integer"
  Minimum="0" Maximum="255">
</des:RangeValidator>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Date

This Date control uses DropDownList segments for month and year. It uses an IntegerTextSegment for the day. It has a DataTypeCheckValidator attached to confirm it uses a valid date.

It uses the hint feature with a label showing a hint for each segment.

Note: The DateTextBox in Peter's Date and Time module is a far more powerful and elegant date field with popup calendar. This is a very lightweight example to demonstrate using DropDownListSegments and a Validator.

```
<des:MultiSegmentDataEntry id="MultiSegmentDataEntry1" runat="server"
   SharedHintFormatterName="LtYellow-Small" >
<Segments>
   <des:DropDownListSegment FormattingTextAfter="/" Hint="Month"
      DisplayTextAfter=" " MaxLength="2">
      <Items>
         <asp:ListItem Value="01">Jan</asp:ListItem>
         <asp:ListItem Value="02">Feb</asp:ListItem>
         <asp:ListItem Value="03">Mar</asp:ListItem>
         <asp:ListItem Value="04">Apr</asp:ListItem>
         <asp:ListItem Value="05">May</asp:ListItem>
         <asp:ListItem Value="06">Jun</asp:ListItem>
         <asp:ListItem Value="07">Jul</asp:ListItem>
         <asp:ListItem Value="08">Aug</asp:ListItem>
         <asp:ListItem Value="09">Sep</asp:ListItem>
         <asp:ListItem Value="10">Oct</asp:ListItem>
         <asp:ListItem Value="11">Nov</asp:ListItem>
         <asp:ListItem Value="12">Dec</asp:ListItem>
      </Items>
   </des:DropDownListSegment>
   <des:IntegerTextSegment FormattingTextAfter="/"
      AutoWidth="False" Width="20px"
      FillLeadZeros="2"  DisplayTextAfter=" " MaxLength="2"
      ShowSpinner="True" SpinnerMinValue="1" SpinnerMaxValue="31"
      Hint="Day. Between 1-31" ></des:IntegerTextSegment>
   <des:DropDownListSegment Hint="Year">
      <Items>
         <asp:ListItem Value="1999">1999</asp:ListItem>
         <asp:ListItem Value="2000">2000</asp:ListItem>
         <asp:ListItem Value="2001">2001</asp:ListItem>
         <asp:ListItem Value="2002">2002</asp:ListItem>
         <asp:ListItem Value="2003">2003</asp:ListItem>
         <asp:ListItem Value="2004">2004</asp:ListItem>
      </Items>
   </des:DropDownListSegment>
</Segments>
</des:MultiSegmentDataEntry>
<asp:Label id="HintLabel1" runat="server"></asp:Label>
<des:DataTypeCheckValidator id="DataTypeCheckValidator1" runat="server"
   ControlIDToEvaluate="MultiSegmentDataEntry1"
   ErrorMessage="Invalid date: {TEXTVALUE}" DataType="Date">
</des:DataTypeCheckValidator>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

MultiSegmentDataEntryValidator Control

Note: If you are using the Native Validation Framework, there is another MultiSegmentDataEntryValidator. See "MultiSegmentDataEntryValidator Control".

Condition: MultiSegmentDataEntryCondition

Supported controls: MultiSegmentDataEntry

Can evaluate blank fields: Determined by the IgnoreBlankText property

The MultiSegmentDataEntryValidator confirms that the segments of the MultiSegmentDataEntry control match the rules you specified on the Segment objects. You should always use this Validator or its Condition on each MultiSegmentDataEntry control.

This Validator uses DES's Validation framework. It has the same properties and expects to work with DES's ValidationSummary and other Validation framework features. See the **Validation User's Guide** for details of DES Validators.

Using This Condition

Specify the MultiSegmentDataEntry control with the **ControlIDToEvaluate** property.

Example

Determine if MultiSegmentDataEntry1 matches its rules for a phone number.

```
<des:MultiSegmentDataEntryValidator
   id=MultiSegmentDataEntryValidator1 runat="server"
   ControlIDToEvaluate="MultiSegmentDataEntry1"
   ErrorMessage="A phone number requires this format: ### ### ####">
</des:MultiSegmentDataEntryValidator >
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Condition Properties for MultiSegmentDataEntryValidator

Each Validator control includes a Condition that has several properties that require setup. The Properties Editor shows them in the "Condition" section. For all properties not shown here, see the **Validation User's Guide**.

The following list are properties specific to this Condition:

- **ControlIDToEvaluate** (string) – Identifies the MultiSegmentDataEntry control that will be evaluated. This property takes the ID of the control.

  An exception is thrown at runtime when this is blank, unknown, not in the same or ancestor naming container, is Visible=false, or a control class that is not supported.

- **ControlToEvaluate** (System.Web.UI.Control) – An alternative to **ControlIDToEvaluate**. Use it when the MultiSegmentDataEntry control is not in the same or ancestor naming container. It must be assigned programmatically. For example, if you have a Validator instance in the variable "Val1" and a MultiSegmentDataEntry instance in the variable "MultiSegmentDataEntry1", write code like this: `Val1.ControlToEvaluate = MultiSegmentDataEntry1`.

  When programmatically assigning properties to a Validator control or Condition class, if you have access to the control object that will be evaluated, it is better to assign it here than assign its ID to the **ControlIDToEvaluate** property because DES operates faster using **ControlToEvaluate**.

- **IgnoreBlankText** (Boolean) – Determines how the Validator evaluates when all segments are blank.

  When `true`, the Condition cannot evaluate. You use a RequiredTextValidator to report errors on it.

  When `false`, it evaluates as "failed", which reports an error.

  It defaults to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Subclassing MultiSegmentDataEntry

If you subclass `PeterBlum.DES.MultiSegmentDataEntry`, you must tell DES about it so it can be used with DES's validators. Here's how.

1. Open the **custom.des.config** file in your **[web application]\DES** folder.
2. Locate the `<ThirdPartryControls>` section.
3. Add this node to `<ThirdPartyControls>` where *class* is your full class name.

```
<ThirdPartyControl class="class" sameas="textbox" property="ValidatorText" >
   <GetTextScript>[DES_GetTextMSDE]</GetTextScript>
   <GetChildHUScript>[DES_GetChildMSDE]<GetChildHUScript>
</ThirdPartyControl>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Additional Topics for Using These Controls

This section covers a variety of special cases when using these controls.

Click on any of these topics to jump to them:

- Page Level Properties
- Validation with the Native Validation Framework
- JavaScript Support Functions
- Adding Your JavaScript to the Page
- Troubleshooting

These topics are found in the General Settings Guide:

- Using these Controls with AJAX
- The ViewState and Preserving Properties for PostBack
- Establishing Default Localization for the Web Form
- Using Style Sheets
- The String Lookup System
- The Global Settings Editor
- Using Server Transfer and Using Alternative HttpHandlers
- Using a Redistribution License
- Browser Support and The TrueBrowser Class

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Page Level Properties

The **PeterBlum.DES.Globals.Page** object contains several properties that affect all controls on the page. They include setting the CultureInfo object, getting the Browser details, and enabling JavaScript.

The **Page** property on **PeterBlum.DES.Globals** uses the class `PeterBlum.DES.DESPage`. When accessed through **PeterBlum.DES.Globals.Page**, you will have an object that is unique to the current thread. It is really a companion to the Page object of a web form, hosting details related to DES. Properties set on it will not affect any other request for a page.

Many of its properties can be set by using the PageManager control. See the **General Features Guide**.

Properties on PeterBlum.DES.Globals.Page

You generally assign properties to **PeterBlum.DES.Globals.Page** in your `Page_Init()` or `Page_Load()` method. Your post back event handler methods can also assign properties.

- **CultureInfo** (System.Globalization.CultureInfo) – Cultures define date, time, number and text formatting for a program to follow. DES uses this value within its data types (`PeterBlum.DES.DESTypeConverter` classes) as it translates between strings and values. For example, the Date data type uses this to get the DateTimeFormatInfo class, which defines the short date pattern (ex: MM/dd/yyyy) and date separator. The Currency data types use the NumberFormatInfo class to get the currency symbol, decimal symbol, and number of decimal places.

  The **CultureInfo** property uses CultureInfo.CurrentCulture by default. This value is determined by the web server's .Net settings, the web.config's `<globalization>` tag, or the <% @Page %> tag with the **Culture** property.

  Web.Config setting – Affects the entire site

  ```
  <globalization Culture="en-US" [other properties] />
  ```

  Page Setting – Affects a page

  ```
  <%@Page Culture="en-US" [other page properties] %>
  ```

  You can set it programmatically in your `Page_Init()` method or in the `Application_BeginRequest()` method of **Global.asax**. Use the .Net Framework method `CultureInfo.CreateSpecificCulture()`. For example, assigning the US culture looks like this:

  ```
  PeterBlum.DES.Globals.Page.CultureInfo =
     CultureInfo.CreateSpecificCulture("en-US")
  ```

  Changing the properties of CultureInfo programmatically

  Assign values to **PeterBlum.DES.Globals.Page.CultureInfo**. Here are some examples:

  [C#]

  ```
  System.Globalization.NumberFormatInfo vNFI =
     PeterBlum.DES.Globals.Page.CultureInfo.NumberFormat;
  vNFI.DecimalSeparator = ".";
  vNFI.CurrencySymbol = "€";
  ```

  [VB]

  ```
  Dim vNFI As System.Globalization.NumberFormatInfo = _
     PeterBlum.DES.Globals.Page.CultureInfo.NumberFormat
  vNFI.DecimalSeparator = "."
  vNFI.CurrencySymbol = "€"
  ```

- **Browser** (PeterBlum.DES.TrueBrowser) – Detects the actual browser that is requesting the page and configures the HTML and JavaScript code returned to work with that browser. If the browser doesn't support the client-side scripting code for filtering keystrokes, the **SupportsKeyboardFiltering** property is `false` and these textboxes do not generate most of their client-side scripts. See "Browser Support" in the **General Features Guide**.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **InitialFocusControl** (System.Web.UI.Control) – Sets the focus on the page to this control when the page is first loaded. Assign this property to a reference to the control that should get the initial focus. If the control is hidden or disabled, focus will not be set because browsers do not permit it.

  Typically this is set within `Page_Load()` or a post back event handler.

  When `null/nothing`, no field gets initial focus. It defaults to `null/nothing`.

  Example

  Set focus to a textbox associated with TextBox1:

  [C#]

  ```
  PeterBlum.DES.Globals.Page.InitialFocusControl = TextBox1;
  ```

  [VB]

  ```
  PeterBlum.DES.Globals.Page.InitialFocusControl = TextBox1
  ```

  License Note: This property requires a license for the Peter's Interactive Pages.

- **JavaScriptEnabled** (Boolean) – Determines if the browser really has JavaScript enabled. It automatically detects if JavaScript is enabled after the first post back for a session. Prior to that first post back, it is `true`. After that, it is `true` when JavaScript is enabled and `false` when it is not.

  When `false`, the page will be generated as if the browser does not support JavaScript. No controls will output JavaScript and may draw themselves differently, knowing that a client-side only feature that doesn't work is inappropriate to output. For example, spinners on textboxes are not included. The server side will handle these controls gracefully on post back.

  This feature stores its state in the Session collection. If the Session is not working or has been cleared, it will reset to `true` and attempt to resolve the JavaScript state on the next post back.

  If you do not want this detection feature enabled, set **DetectJavaScript** to `false`.

  You can set this value directly in `Page_Load()`. It lets you turn off all of DES's JavaScript features on demand. For example, your customers can identify if they use JavaScript on their browser in a configuration screen. It only affects the current page so set it on each page where needed.

- **DetectJavaScript** (Boolean) – When `true`, the **JavaScriptEnabled** property will monitor for JavaScript support. When `false`, it will not. It defaults to the global **DefaultDetectJavaScript** property, which defaults to `true`. You set **DefaultDetectJavaScript** with the **Global Settings Editor**.

- **ValueWhenBlankMode** (enum PeterBlum.DES.ValueWhenBlankMode) – When using the **ValueWhenBlank** and **ValueWhenBlankCssClass** properties on any TextBox, this determines how setting focus to the textbox modifies the appearance of the control. The enumerated type `PeterBlum.DES.ValueWhenBlankMode` has these values:

  - `RetainBoth` – No changes are made. If **ValueWhenBlank** and **ValueWhenBlankCssClass** are in use, they will remain in use while focus is on the field. The user will have to manually remove the current text.
  - `RemoveText` – The text will change to "" if it is set to **ValueWhenBlank**. The style sheet class will not be changed.
  - `RemoveBoth` – The text will change to "" if it is set to **ValueWhenBlank**. The style sheet class will be assigned to the original style sheet class name. This is the default.

  It defaults to the global **DefaultValueWhenBlankMode** property, which defaults to `RemoveBoth`. You set **DefaultValueWhenBlankMode** with the **Global Settings Editor**.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **EnableButtonImageEffects** (enum PeterBlum.DES.EnableButtonImageEffects) – Many buttons can show up to 3 images: normal, pressed, and mouseover. By default, these effects are set up based on the presence of the actual files. However, DES cannot always see the files are present. For example, the URL uses `http://`. **EnableButtonImageEffects** lets you to specify that the images are present or not.

  The enumerated type has these values:

  - `None` - Never use image effects.
  - `Always` - Always use image effects. Assume that all image files are available
  - `Auto` - Detect the files, if possible before using them
  - `Pressed` - Always set up for pressed. Never set up for mouse over
  - `MouseOver` - Always set up for mouseover. Never set up for pressed

  It defaults to `EnableButtonImageEffects.Auto`.

- **PageIsLoadingMsg** (string) – The error message to display on the client-side if the user interacts with this control before it is initialized. It defaults to "`Page is loading. Please wait.`".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

SpinnerManager Property

The **PeterBlum.DES.Globals.Page** class and PageManager control offer the **SpinnerManager** property to customize the look and behavior of spinners found on time and numeric textboxes throughout DES. Each of these properties has a default that is set within the **Global Settings Editor** in the "SpinnerManager Defaults" section.

- **SpinnerManager.IncrementButtonUrl** (string) – The up arrow buttons shown in spinners (used by the time and numeric textboxes throughout DES.) This string must be assigned to a URL to an image representing the concept "Next". It's used by both the Next Minutes and Next Hours buttons.

  It defaults to value of the **DefaultIncrementButtonUrl** property which is set in the **Global Settings Editor** and defaults to `"{APPEARANCE}/Shared/UpArrow1.gif"` (▲).

  The `<img>` tag uses the text "+" for the `alt=` attribute. (It cannot be customized.)

  #### Special Symbols for URLs

  The "{APPEARANCE}" token will be replaced by the default path to the **Appearance** folder, which you defined as you set up the web site.

  Supports the use of the tilde (~) as the first character to be replaced by the virtual path to the web application.

  #### Images for Pressed and MouseOver Effects

  You can have images for pressed and mouseover effects as well as the normal image. The names of the image files determine their purpose. Define the name of the normal image. For example, "myimage.gif". Create the pressed version and give it the same name, with "Pressed" added before the extension. For example, "myimage*pressed*.gif". Create the mouseover version and give it the same name, with "MouseOver" added before the extension. For example, myimage*mouseover*.gif.

  The **IncrementButtonUrl** property should refer to the normal image. DES will detect the presence of the other two files. If any are missing, DES continues to use the normal image for that case. *Note: Auto detection only works when the URL is a virtual path to a file. You can manage this capability with the PeterBlum.DES.Globals.Page.EnableButtonImageEffects.*

  If you need more control over paths for pressed and mouseover images, you can embed up to 3 URLs into this property using a pipe (|) delimited list. The order is important: `normal|pressed|mouseover`. If you want to omit the pressed image, use: `normal||mouseover`. If you want to omit the mouseover image, use: `normal|pressed`.

- **SpinnerManager.DecrementButtonUrl** (string) – The down arrow buttons shown in spinners (used by the time and numeric textboxes throughout DES.) This string must be assigned to a URL to an image representing the concept "Previous". It's used by both the Previous Minutes and Previous Hours buttons.

  It defaults to value of the **DefaultDecrementButtonUrl** property which is set in the **Global Settings Editor** and defaults to `"{APPEARANCE}/Shared/DnArrow1.gif"` (▼).

  The `<img>` tag uses the text "-" for the `alt=` attribute. (It cannot be customized.)

  #### Special Symbols for URLs

  The "{APPEARANCE}" token will be replaced by the default path to the **Appearance** folder, which you defined as you set up the web site.

  Supports the use of the tilde (~) as the first character to be replaced by the virtual path to the web application.

  #### Images for Pressed and MouseOver Effects

  You can have images for pressed and mouseover effects as well as the normal image. The names of the image files determine their purpose. Define the name of the normal image. For example, "myimage.gif". Create the pressed version and give it the same name, with "Pressed" added before the extension. For example, "myimage*pressed*.gif". Create the mouseover version and give it the same name, with "MouseOver" added before the extension. For example, myimage*mouseover*.gif.

  The **DecrementButtonUrl** property should refer to the normal image. DES will detect the presence of the other two files. If any are missing, DES continues to use the normal image for that case. *Note: Auto detection only works when the*

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

URL is a virtual path to a file. You can manage this capability with the PeterBlum.DES.Globals.Page.EnableButtonImageEffects.

If you need more control over paths for pressed and mouseover images, you can embed up to 3 URLs into this property using a pipe (|) delimited list. The order is important: `normal|pressed|mouseover`. If you want to omit the pressed image, use: `normal||mouseover`. If you want to omit the mouseover image, use: `normal|pressed`.

- **SpinnerManager.AutoRepeatSpeed1** (int) – The number of milliseconds to wait between each change to the textbox's value while the user holds the mouse down. This time is used for the first 5 command executions. **AutoRepeatSpeed2** is used after that.

  It defaults to value of the **DefaultAutoRepeatSpeed1** property which is set in the **Global Settings Editor** and defaults to 500 (.5 seconds).

- **SpinnerManager.AutoRepeatSpeed2** (int) – The number of milliseconds to wait between each change to the textbox's value while the user holds the mouse down. This time is used after the first 5 command executions. **AutoRepeatSpeed1** is used before that.

  It defaults to value of the **DefaultAutoRepeatSpeed2** property which is set in the **Global Settings Editor** and defaults to 250 (.25 seconds).

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Validation with the Native Validation Framework

DES provides its own validator controls to work with the Integer, Decimal, Currency, and Percent textboxes. They are used in place of the native CompareValidator and RangeValidator, and provide equivalent functionality, but can handle the differences in number formats that are not part of the original validator controls. They automatically configure themselves based on the textbox's property so you don't have to set properties like **Type**, **AllowsNegatives**, **ShowThousandsSeparators**, and **ShowCurrencySymbol**.

Each numeric textbox must have a CompareValidator attached that will verify its contents are legal. While these textboxes may appear to work without it, always plan for users who don't have client-side validation support on their browsers (or hackers, who turn off javascript in hopes that your server side code doesn't block their illegal inputs.)

Click on any of these topics to jump to them:

- Setting Up DES with the Native Validation Framework
- Check the DataType with the CompareValidator Control
- Compare To Value with the CompareValidator Control
- Compare Two Fields with the CompareValidator Control
- RangeValidator Control
- DifferenceValidator Control
- MultiSegmentDataEntryValidator

You will continue to use the native ASP.NET validators in many cases:

- RequiredFieldValidator – All DES textboxes and the MultiSegmentDataEntry control use it to determine if the control is blank.
- CompareValidator – Appropriate for DES's TextBox, FilteredTextBox, and MultiSegmentDataEntry control.
- RangeValidator – Appropriate for DES's TextBox, FilteredTextBox, and MultiSegmentDataEntry control.
- RegularExpressionValidator – Appropriate for DES's TextBox, FilteredTextBox, and MultiSegmentDataEntry control. It works with the numeric textboxes although it's unlikely you need it with them.
- CustomValidator – All DES textboxes and the MultiSegmentDataEntry control use it to determine if the control is blank.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Setting Up DES with the Native Validation Framework

DES's Validator controls for the Native Validation Framework are in the **PeterBlum.DES.NativeValidators.dll** assembly, which is in your **\bin** folder. *It was automatically added when you ran the Web Application Updater on your web application.*

You may want to take these actions before adding these validators to your webform:

- Add the **PeterBlum.DES.NativeValidators** assembly to the Visual Studio or Visual Web Developer toolbox. See "Adding To The Visual Studio/Visual Web Developer Toolbox" in the **Installation Guide**.
- ASP.NET 1.x users should add this line to the top of the webform:

```
<% @Register tagPrefix="desmsval" namespace="PeterBlum.DES.NativeValidators"
   assembly="PeterBlum.DES.NativeValidators" %>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Check the DataType with the CompareValidator Control

Applies to Integer, Decimal, Currency, and Percent TextBoxes. For others, use the original ASP.NET CompareValidator

The CompareValidator can check that the number entered exactly matches the formatting rules on the textbox when you set its **Operator** property to `DataTypeCheck`. Always add this validator to these textboxes. It will automatically determine the exact formatting rules from the textbox.

Using this Validator

Set the ID of the textbox in **ControlToValidate**. Set the **Operator** property to `DataTypeCheck`.

Properties

The CompareValidator from **PeterBlum.DES.NativeValidators** is very similar to its counterpart that comes with ASP.NET. Here are its most important properties. Those not shown below are documented here:

CompareValidator Members

- **ControlToValidate** (string) – The ID of the control to evaluate. It can be any DES textbox except an Enhanced TextBox or FilteredTextBox. The TextBox must be in the same Naming Container as the validator. *This is a limitation of the Native Validation Framework. Switch to DES Validation Framework to overcome this limitation.*
- **Operator** (enum System.Web.UI.WebControls.ValidatorCompareOperator) – Always set this to `DataTypeCheck` when using this validator to check for a legal number.
- **DataType** (enum PeterBlum.DES.NativeValidators.DataTypeMode) – Leave this set to `Auto`. *It is only used with the TimeOfDayTextBox.*

Note: If you have two or more of these validators on one control, consider setting their Display property to Dynamic.

Example: IntegerTextBox

```
<des:IntegerTextBox id="IntegerTextBox1" runat="server" />

<desmsval:CompareValidator id="CompareValidator1" runat="server"
     ControlIDToEvaluate="IntegerTextBox1"
     ErrorMessage="Incorrect Integer value."
     Operator="DataTypeCheck" />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Compare To Value with the CompareValidator Control

Applies to Integer, Decimal, Currency, and Percent TextBoxes. For others, use the original ASP.NET CompareValidator

Compare the value of the textbox to a number in the **ValueToCompare** property. It only evaluates when the textbox has a legal value.

Using this Validator

Set the ID of the textbox in **ControlToValidate**. Set the number to compare in **ValueToCompare**. Set the comparison operator in **Operator**.

Properties

The CompareValidator from **PeterBlum.DES.NativeValidators** is very similar to its counterpart that comes with ASP.NET. Here are its most important properties. Those not shown below are documented here:

CompareValidator Members

- **ControlToValidate** (string) – The ID of the control to evaluate. It can be any DES textbox except an Enhanced TextBox or FilteredTextBox. The TextBox must be in the same Naming Container as the validator. *This is a limitation of the Native Validation Framework. Switch to DES Validation Framework to overcome this limitation.*
- **Operator** (enum System.Web.UI.WebControls.ValidatorCompareOperator) – The operator for the comparison between **ControlToValidate** and **ValueToCompare**. Use any value from this enumerated type except `DataTypeCheck`.
  - `Equal`
  - `NotEqual`
  - `GreaterThan`
  - `GreaterThanEqual`
  - `LessThan`
  - `LessThanEqual`
- **DataType** (enum PeterBlum.DES.NativeValidators.DataTypeMode) – Leave this set to `Auto`. *It is only used with the TimeOfDayTextBox.*
- **ValueToCompare** (string) – When using an IntegerTextBox, it must be a string representing an integer. For others, can be either a string representing a decimal or integer. When programmatically assigning this value and you have an integer or decimal variable, use the `ToString()` method on that variable.

Note: If you have two or more of these validators on one control, consider setting their Display property to Dynamic.

Example: IntegerTextBox

```
<des:IntegerTextBox id="IntegerTextBox1" runat="server" />

<desmsval:CompareValidator id="CompareValidator1" runat="server"
     ControlIDToEvaluate="IntegerTextBox1" Operator="LessThan"
     ValueToCompare="10"
     ErrorMessage="Must be less than 10."
     />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Compare Two Fields with the CompareValidator Control

Applies to Integer, Decimal, Currency, and Percent TextBoxes. For others, use the original ASP.NET CompareValidator

Compare the value of one textbox to a value of another.

This is commonly used when developing a range where the first textbox must be less than or equal to the second.

Using this Validator

Set the ID of the two textboxes in **ControlToValidate** and **ControlToCompare**. Set the comparison operator in **Operator**.

Properties

The CompareValidator from **PeterBlum.DES.NativeValidators** is very similar to its counterpart that comes with ASP.NET. Here are its most important properties. Those not shown below are documented here:

CompareValidator Members

- **ControlToValidate** (string) – The ID of the control to evaluate. It can be any DES textbox except an Enhanced TextBox or FilteredTextBox. The TextBox must be in the same Naming Container as the validator. *This is a limitation of the Native Validation Framework. Switch to DES Validation Framework to overcome this limitation.*
- **Operator** (enum System.Web.UI.WebControls.ValidatorCompareOperator) – The operator for the comparison between **ControlToValidate** and **ControlToCompare**. Use any value from this enumerated type except `DataTypeCheck`.
  - `Equal`
  - `NotEqual`
  - `GreaterThan`
  - `GreaterThanEqual`
  - `LessThan`
  - `LessThanEqual`
- **DataType** (enum PeterBlum.DES.NativeValidators.DataTypeMode) – Leave this set to `Auto`. *It is only used with the TimeOfDayTextBox.*
- **ControlToCompare** (string) – The ID from the second control to evaluate. If the ControlToValidate, it must be an IntegerTextBox. If the ControlToValidate is a Decimal, Currency, or Percent TextBox, this can be any of those control types. The TextBox must be in the same Naming Container as the validator.

Note: If you have two or more of these validators on one control, consider setting their Display property to Dynamic.

Example: Two IntegerTextBoxes

```
<des:IntegerTextBox id="StartInteger" runat="server" />
<des:IntegerTextBox id="EndInteger" runat="server" />

<desmsval:CompareValidator id="CompareValidator1" runat="server"
     ControlToValiInteger="StartInteger" ControlToCompare="EndInteger"
     Operator="LessThan"
     ErrorMessage="Start Integer must be less than the End Integer."
     />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

RangeValidator Control

Applies to Integer, Decimal, Currency, and Percent TextBoxes. For others, use the original ASP.NET CompareValidator

Confirm the textbox value is within the range established by their minimum and maximum values.

Using this Validator

It automatically uses the TextBox's **MinValue** and **MaxValue** properties to determine the range. If you leave those properties unassigned, you can still set their values directly in the RangeValidator's **Minimum** and **Maximum** properties.

Set the ID of the textbox in **ControlToValidate**. Set the minimum and maximum in either the TextBox's **MinValue** and **MaxValue** or in the RangeValidator's **Minimum** and **Maximum** properties.

Properties

The RangeValidator from **PeterBlum.DES.NativeValidators** is very similar to its counterpart that comes with ASP.NET. Here are its most important properties. Those not shown below are documented here:

RangeValidator Members

- **ControlToValidate** (string) – The ID of the control to evaluate. It can be any DES textbox except an Enhanced TextBox or FilteredTextBox. The TextBox must be in the same Naming Container as the validator. *This is a limitation of the Native Validation Framework. Switch to DES Validation Framework to overcome this limitation.*
- **DataType** (enum PeterBlum.DES.NativeValidators.DataTypeMode) – Leave this set to `Auto`. *It is only used with the TimeOfDayTextBox.*
- **Minimum** (string) – Used when the associated textbox's **MinValue** property has not been set. When using an IntegerTextBox, it must be a string representing an integer. For others, can be either a string representing a decimal or integer. When programmatically assigning this value and you have an integer or decimal variable, use the `ToString()` method on that variable.
- **Maximum** (string) – Used when the associated textbox's **MaxValue** property has not been set. When using an IntegerTextBox, it must be a string representing an integer. For others, can be either a string representing a decimal or integer. When programmatically assigning this value and you have an integer or decimal variable, use the `ToString()` method on that variable.

Note: If you have two or more of these validators on one control, consider setting their Display property to Dynamic.

Example: IntegerTextBox

```
<des:IntegerTextBox id="IntegerTextBox1" runat="server"
     MinInteger="1" MaxInteger="100" />

<desmsval:RangeValidator id="RangeValidator1" runat="server"
     ControlToValiInteger="IntegerTextBox1"
     ErrorMessage="Enter a number between {MINIMUM} and {MAXIMUM}."
     />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

DifferenceValidator Control

Applies to Integer, Decimal, Currency, and Percent TextBoxes.

Use this validator to enforce that two textboxes values are a certain number apart. For example, you want two IntegerTextBoxes to be no closer than 5 in value or two DecimalTextBoxes to be no more than 0.5 apart.

Using this Validator

Set the ID of the two textboxes in **ControlToValidate** and **ControlToCompare**. Set the difference in **DifferenceValue**. Set the comparison operator in **Operator**.

Properties

The CompareValidator from **PeterBlum.DES.NativeValidators** is very similar to the CompareValidator that comes with ASP.NET, expanding upon its ability to compare two controls. Here are its most important properties. Those not shown below are documented here:

CompareValidator Members

- **ControlToValidate** (string) – The ID of the control to evaluate. It can be any DES textbox except an Enhanced TextBox or FilteredTextBox. The TextBox must be in the same Naming Container as the validator. *This is a limitation of the Native Validation Framework. Switch to DES Validation Framework to overcome this limitation.*
- **Operator** (enum System.Web.UI.WebControls.ValidatorCompareOperator) – The operator for the comparison between **ControlToValidate** and **ControlToCompare**. Use any value from this enumerated type except `DataTypeCheck`.
  - `Equal`
  - `NotEqual`
  - `GreaterThan`
  - `GreaterThanEqual`
  - `LessThan`
  - `LessThanEqual`
- **DataType** (enum PeterBlum.DES.NativeValidators.DataTypeMode) – Leave this set to `Auto`. *It is only used with the TimeOfDayTextBox.*
- **ControlToCompare** (string) – The ID from the second control to evaluate. If the ControlToValidate, it must be an IntegerTextBox. If the ControlToValidate is a Decimal, Currency, or Percent TextBox, this can be any of those control types. The TextBox must be in the same Naming Container as the validator.
- **DifferenceValue** (double) - The value to be compared to the difference between the two values. (The difference is always an absolute value.)

Note: If you have two or more of these validators on one control, consider setting their Display property to Dynamic.

Example: Two IntegerTextBoxes that must be no less than 5 apart

```
<des:IntegerTextBox id="StartInteger" runat="server" />
<des:IntegerTextBox id="EndInteger" runat="server" />

<desmsval:DifferenceValidator id="DifferenceValidator1" runat="server"
     ControlToValiInteger="StartInteger" ControlToCompare="EndInteger"
     Operator="GreaterThanEqual" DifferenceValue="5"
     ErrorMessage="The Integers must not be more than 5 days apart."
     />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Example: Two DecimalTextBoxes that must be no more than 0.5 apart

```
<des:DecimalTextBox id="StartDecimal" runat="server" />
<des:DecimalTextBox id="EndDecimal" runat="server" />

<desmsval:DifferenceValidator id="DifferenceValidator1" runat="server"
     ControlToValiDecimal="StartDecimal" ControlToCompare="EndDecimal"
     Operator="LessThanEqual" DifferenceValue="0.5"
     ErrorMessage="The Decimals must not be more than 5 days apart."
     />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

MultiSegmentDataEntryValidator Control

Note: If you are using the DES Validation Framework, it includes a different MultiSegmentDataEntryValidator. See "MultiSegmentDataEntryValidator Control".

The MultiSegmentDataEntryValidator confirms that the segments of the MultiSegmentDataEntry control match the rules you specified on the Segment objects. You should always use this Validator or its Condition on each MultiSegmentDataEntry control.

Using This Validator

Specify the MultiSegmentDataEntry control with the **ControlToValidate** property. If you want this validator to work like a RequiredFieldValidator, reporting an error when all segments are blank, set **IgnoreBlankText** to `false`.

Properties for MultiSegmentDataEntryValidator

- **ControlToValidate** (string) – The ID of the MultiSegmentDataEntry control. It must be in the same Naming Container as the validator. *This is a limitation of the Native Validation Framework. Switch to DES Validation Framework to overcome this limitation.*
- **IgnoreBlankText** (Boolean) – Determines how the Validator evaluates when all segments are blank.

  When `true`, the validator cannot evaluate. You use a RequiredFieldValidator to report errors on it.

  When `false`, it reports an error.

  It defaults to `true`.

Example

Determine if MultiSegmentDataEntry1 matches its rules for a phone number.

```
<desmsval:MultiSegmentDataEntryValidator
   id="MultiSegmentDataEntryValidator1" runat="server"
   ControlToValidate="MultiSegmentDataEntry1"
   ErrorMessage="A phone number requires this format: ### ### ####">
</desmsval:MultiSegmentDataEntryValidator >
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

JavaScript Support Functions

This section shows how to communicate with these controls from your own JavaScript.

DES supplies the following client-side functions to any page that includes these controls.

Click on any of these topics to jump to them:

- General Utilities
- Getting and Setting the Value of Numeric TextBoxes
- Validation Functions
- MultiSegmentDataEntry Functions

General Utilities

function DES_GetById(pID)

Returns the DHTML element associated with the ID supplied. This is a wrapper around the functions `document.all[]` and `document.GetElementById()` so that you can get the field using browser independent code.

Parameters

pID

The **ClientID** property value from the server side control. It is the value written into the `id=` attribute of the HTML element. See "Embedding the ClientID into your Script".

Return value

Returns the field object or null.

Example

```
var vOtherField = DES_GetById('DateTextBox1');
```

function DES_ParseInt(pText)

It converts it into an integer number and returns the number. While the JavaScript `parseInt()` function is supposed to do this, when there is a lead zero, `parseInt()` believes the number is octal (base 8). Thus, 08 is returned as 10. Dates often have lead zeros. So call this instead of `parseInt()`. *Internally, it calls parseInt() after stripping off the lead zeroes.*

Parameters

pText

The string to convert to an integer.

Return value

An integer. If the text represented a decimal value, it will return the integer portion. If it cannot be converted, it returns `NaN` which you can detect with the JavaScript function `isNaN(value)`.

Example

```
var vNumber = DES_ParseInt("03");   // returns 3
if (!isNaN(vNumber))
   // do something with vNumber
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

function DES_SetFocus(pID)

Sets focus to the HTML element whose ID is passed in. It will not set focus if the element is not present or it's illegal to set focus (such as its invisible). It will also select the contents of a textbox, if the ID is to a textbox.

It calls your custom focus function defined in **PeterBlum.DES.Globals.Page.SetFocusFunctionName** to assist it to setting focus. (See "Properties on PeterBlum.DES.Globals.Page" in the **General Features Guide**.)

Parameters

pID

The **ClientID** property value from the server side control. It is the value written into the `id=` attribute of the HTML element. See "Embedding the ClientID into your Script".

Return value

Returns the field object or null.

Example

```
DES_SetFocus('DateTextBox1');
```

function DES_Round(pValue, pMode, pDecimalPlaces)

Rounds a decimal value in several ways.

Parameters

pValue

The initial decimal value.

pMode

An integer representing one of the rounding modes:

0 = Truncate – Drop the decimals after pDecimalPlaces

1 = Currency – Round to the nearest even number

2 = Point5 – Round to the next number if .5 or higher; round down otherwise

3 = Ceiling – Returns the smallest integer greater than or equal to a number. When it's a negative number, it will return the number closest to zero.

4 = NextWhole - Returns the smallest integer greater than or equal to a number. When it's a negative number, it will return the number farthest from zero.

pDecimalPlaces

The number of decimal places to preserve. For example, when 2, it rounds based on the digits after the 2nd decimal place.

Return value

Returns the rounded decimal value.

Example

```
var PI = 3.14159;
var vResult = DES_Round(PI, 0, 0);  // Truncate: returns 3
vResult = DES_Round(PI, 1, 2);      // Currency: returns 3.14
vResult = DES_Round(PI, 3, 0);      // Ceiling: returns 4
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

function DES_Trunc(pValue)

Returns the integer part of a decimal value. Converts the type from float to integer.

Parameters

pValue

The initial decimal value.

Return value

Returns the integer part of a decimal value. Converts the type from float to integer.

Example

```
var PI = 3.14159;
var vResult = DES_Trunc(PI);  // returns 3
```

function DES_SetInnerHTML(pField, pHTML)

A browser independent way to update the inner HTML of a tag. Usually you will define a `<span>` tag with an ID. The inner HTML of that tag will be updated. A `System.Web.UI.WebControls.Label` creates such a `<span>` tag and its **ClientID** is the ID to find the tag on the page.

Parameters

pField

The DHTML element for the HTML table. Use `DES_GetById()` to convert a **ClientID** into an DHTML element. See "Embedding the ClientID into your Script".

pHTML

The inner HTML.

Example

```
DES_SetInnerHTML(DES_GetById('Label1'), 'New Text');
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Getting and Setting the Value of Numeric TextBoxes

function DES_GetDTTBValue(pID)

Supports Integer, Decimal, Currency and Percent TextBoxes.

Retrieve the value of the numeric textbox. When using an IntegerTextBox, the value returned is an integer. Otherwise it is a floating point. If it cannot convert the text into a number, it returns `null`. *The same function is used by all date, time and numeric textboxes. The "DTTB" stands for "DataTypeTextBox".*

Parameters

pID

The **ClientID** property value from the server side control. It is the value written into the `id=` attribute of the HTML element. See "Embedding the ClientID into your Script".

Return value

Returns an integer or floating point value. If it cannot convert the text into a valid number, it returns `null`.

Example

```
var vNumber = DES_GetDTTBValue('IntegerTextBox1');
if (vNumber != null)
   // do something with vNumber
```

function DES_SetDTTBValue(pID, pValue, pAfter)

Supports Integer, Decimal, Currency and Percent TextBoxes.

Assign a number to a numeric TextBox. *The same function is used by all date, time and numeric textboxes. The "DTTB" stands for "DataTypeTextBox".*

Parameters

pID

The **ClientID** property value from the server side control. It is the value written into the `id=` attribute of the HTML element. See "Embedding the ClientID into your Script".

pValue

The number. When using an IntegerTextBox, the value must be an integer type. Otherwise pass a floating point value.

If you pass `null`, it clears the textbox (assigning it to the **ValueWhenBlank** if setup).

pAfter

(optional) Determines what else happens. It can call your function defined in **OnChangeFunctionName**. It can fire the DHTML onchange event (which will update validators and cause any DES control that is monitoring the TextBox to fire.)

It takes these integers:

0 - Do nothing.

1 - Fire OnChangeFunction but do not call onchange event.

2 - Fire onchange event but do not call OnChangeFunction.

4 or `null` - fire onchange event and call OnChangeFunction.

10 - fire onchange event and call OnChangeFunction when **CommandsFireOnChangeFunction** is `true`.

`null` is usually passed unless another one of these settings make more sense.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Example

```
DTB_SetDTTBValue('IntegerTextBox1', 100, null);
```

If you want to clear the numeric TextBox, pass `null` into the *pValue* parameter.

```
DTB_SetDTTBValue('IntegerTextBox1', null, null);
```

function DES_FormatDTTBValue(pID, pValue)

Supports Integer, Decimal, Currency and Percent TextBoxes.

Prepares a string from the value supplied, formatting it according to the rules of the TextBox control. Use this to create strings to place in label fields.

Parameters

pID

The **ClientID** property value from the server side control. It is the value written into the `id=` attribute of the HTML element. See "Embedding the ClientID into your Script".

pValue

The number to format. When using an IntegerTextBox, the value must be an integer type. Otherwise pass a floating point value.

If you pass `null`, it clears the textbox (assigning it to the **ValueWhenBlank** if setup).

Return value

Returns the string representation of the value if it could be formatted and `null` if it could not (because it could not convert the value or it was an illegal TextBox ID.)

Example

```
var vText = DES_FormatDTTBValue('IntegerTextBox1', 100);
if (vText != null)
  DES_SetInnerHTML(DES_GetById('Label1'), vText);
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Validation Functions

function DES_FieldChanged(pID)

Runs client-side validation on the control using the DES Validation Framework. It runs all Validators attached to the control.

Parameters

pID

The **ClientID** property value from the server side control. It is the value written into the `id=` attribute of the HTML element. See "Embedding the ClientID into your Script".

Example

```
DES_FieldChanged('TextBox1');
```

function DES_IsValid(pID)

Determines if the control is valid using the DES Validation Framework. It does NOT run validation. It merely returns the state from the most recent validation.

Parameters

pID

The **ClientID** property value from the server side control. It is the value written into the `id=` attribute of the HTML element. See "Embedding the ClientID into your Script".

Return value

Returns `true` if all validators are valid.

Returns `false` if at least one validator is invalid.

Returns `null` if pID has no client-side validators attached.

function DES_ValidateGroup(pGroup)

Runs client-side validation on the control using the DES Validation Framework. It runs all Validators attached to the control.

Parameters

pGroup

The Validation Group name. Can be "". Pass "*" to validate every enabled validator regardless of the group.

Return value

`True` when all validators are valid. `False` otherwise.

Example

```
DES_ValidateGroup('');
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

MultiSegmentDataEntry Functions

The following functions are only available on the MultiSegmentDataEntry control.

function DES_GetTextMSDE(FieldID)

Only supports the MultiSegmentDataEntry control.

Returns a string representation of the current value. When the **ValidatorsUse** property is `TextNoSeparators`, it omits the separators from **FormattingTextBefore** and **FormattingTextAfter**. Otherwise, it includes that formatting.

There is no equivalent method to set the value of this control.

Parameters

FieldID

The **ClientID** property value from the server side control. It is the value written into the `id=` attribute of the HTML element. See "Embedding the ClientID into your Script".

Return value

The string representation of the current value. It does not care if the content is invalid. It returns whatever is there.

Example

```
var text = DES_GetTextMSDE('<% =MultiSegmentDataEntry1.ClientID %>');
```

function DES_ValMSDE(FieldID)

Only supports the MultiSegmentDataEntry control.

Fires all Validators associated with the control, including those directly attached to the data entry controls of the segments. It returns `true` if all are valid; `false` if not.

Use this to update the appearance after making changes, such as through `DES_ClearMSDE()` and `DES_RestoreMSDE()`. Also use it to form logic based on the validation state of the control.

Validation automatically runs at the usual times: after a user's edit and when the page is submitted.

Parameters

FieldID

The **ClientID** property value from the server side control. It is the value written into the `id=` attribute of the HTML element. See "Embedding the ClientID into your Script".

Return value

Boolean. When `true`, it is value. When `false`, it is not.

Example

```
if (DES_ValMSDE('<% =MultiSegmentDataEntry1.ClientID %>'))
  // it is valid
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

function DES_ClearMSDE(FieldID)

Only supports the MultiSegmentDataEntry control.

Clears all segments to a blank state. It does not return a value.

Parameters

FieldID

The **ClientID** property value from the server side control. It is the value written into the `id=` attribute of the HTML element. See "Embedding the ClientID into your Script".

Example

```
DES_ClearMSDE('<% =MultiSegmentDataEntry1.ClientID %>');
```

MultiSegmentDataEntry offers the method `GetClearScript()` that returns this code as you create scripts on the server side. It can call `DES_ValMSDE()` for you by setting the *pValidate* parameter to `true`.

[C#]

```
public string GetClearScript(bool pValidate);
```

[VB]

```
Public Function GetClearScript(ByVal pValidate As Boolean) As String
```

Example

Makes the Clear button's onclick event clear the fields while preventing postback (with `return` set to `false`). Your button must have **CausesValidation**=`false`.

[C#]

```
ClearButton.Attributes.Add("onclick",
  MultiSegmentDataEntry1.GetClearScript(true) + "return false;");
```

[VB]

```
ClearButton.Attributes.Add("onclick", _
  MultiSegmentDataEntry1.GetClearScript(True) + "return false;")
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

function DES_RestoreMSDE(FieldID)

Only supports the MultiSegmentDataEntry control.

Restores all segments to their initial value when the page was loaded or to the last call of `DES_SaveMSDE()`. It does not return a value.

It is often used with an undo command, like a Restore button next to this control.

Parameters

FieldID

The **ClientID** property value from the server side control. It is the value written into the `id=` attribute of the HTML element. See "Embedding the ClientID into your Script".

Example

```
DES_RestoreMSDE('<% =MultiSegmentDataEntry1.ClientID %>');
```

MultiSegmentDataEntry offers the method `GetRestoreScript()` that returns this code as you create scripts on the server side. It can call `DES_ValMSDE()` for you by setting the *pValidate* parameter to `true`.

[C#]

```
public string GetRestoreScript(bool pValidate);
```

[VB]

```
Public Function GetRestoreScript(ByVal pValidate As Boolean) As String
```

Example

Makes the Restore button's onclick event Restore the fields while preventing postback (with `return` set to `false`). Your button must have **CausesValidation**=`false`.

[C#]

```
RestoreButton.Attributes.Add("onclick",
   MultiSegmentDataEntry1.GetRestoreScript(true) + "return false;");
```

[VB]

```
RestoreButton.Attributes.Add("onclick", _
   MultiSegmentDataEntry1.GetRestoreScript(True) + "return false;")
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

function DES_SaveMSDE(FieldID)

Only supports the MultiSegmentDataEntry control.

Captures the current values of the MultiSegmentDataEntry for later restoration when using `DES_RestoreMSDE()`. It does not return a value.

It is invoked as the page is first loaded, so `DES_RestoreMSDE()` will be able to restore to the initial page value.

Parameters

FieldID

The **ClientID** property value from the server side control. It is the value written into the `id=` attribute of the HTML element. See "Embedding the ClientID into your Script".

Example

```
DES_SaveMSDE('<% =MultiSegmentDataEntry1.ClientID %>');
```

MultiSegmentDataEntry offers the method `GetSaveScript()` that returns this code as you create scripts on the server side.

[C#]

```
public string GetSaveScript(bool pValidate);
```

[VB]

```
Public Function GetSaveScript(ByVal pValidate As Boolean) As String
```

Example

Makes the Save button's onclick event Save the fields while preventing postback (with `return` set to `false`). Your button must have **CausesValidation**=`false`. *Note: It's not typical to use a button to save. You generally write code that includes this as part of its processes.*

[C#]

```
SaveButton.Attributes.Add("onclick",
   MultiSegmentDataEntry1.GetSaveScript(true) + "return false;");
```

[VB]

```
SaveButton.Attributes.Add("onclick", _
   MultiSegmentDataEntry1.GetSaveScript(True) + "return false;")
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

function DES_EnableMSDE(FieldID, enable)

Only supports the MultiSegmentDataEntry control.

Enables or disables the segments of the MultiSegmentDataEntry control.

Parameters

FieldID

The **ClientID** property value from the server side control. It is the value written into the `id=` attribute of the HTML element. See "Embedding the ClientID into your Script".

enable

Boolean. When true, enable the segments. When false, disable the segments.

Example

```
DES_EnableMSDE('<% =MultiSegmentDataEntry1.ClientID %>', false);
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Adding Your JavaScript to the Page

Some of DES's features allow you to write your own JavaScript. When writing JavaScript, you can put it in three places:

- Directly on the page. It is typically placed before the `<form>` tag. Be sure to enclose it in `<script>` tags like this:

```
<script type='text/javascript' language='javascript' >
<!--
add your function code here
// -->
</script>
```

- In your `Page_Load()` code using the `Page.RegisterClientScriptBlock()` method. You must still include the `<script>` tags in your code:

[C#]

```
uses System.Text;
...
protected void Page_Load(object sender, System.EventArgs e)
{
   StringBuilder vScript = new StringBuilder(2000);
   vScript.Append("<script type='text/javascript' language='javascript' >\n");
   vScript.Append("<!-- \n");
   vScript.Append( add your function code here );
   vScript.Append("// -->\n</script>\n");
   RegisterClientScriptBlock("KeyName", vScript.ToString());
}
```

[VB]

```
Imports System.Text
...
Protected Sub Page_Load(ByVal sender As object, _
   ByVal e As System.EventArgs)

   Dim vScript As StringBuilder = New StringBuilder(2000)
   vScript.Append("<script type='text/javascript' language='javascript' >")
   vScript.Append("<!-- ")
   vScript.Append( add your function code here )
   vScript.Append("// --></script>")
   RegisterClientScriptBlock("KeyName", vScript.ToString())
End Sub
```

- In a separate file, dedicated to JavaScript. This file doesn't need `<script>` tags. Instead, the page needs `<script src= >` tags to load it. The script tags should appear before the `<form>` tag.

```
<script type='text/javascript' language='javascript' src='url to the file' />
```

Embedding the ClientID into your Script

If your scripts are embedded into your web form, you can use this syntax to get the **ClientID**:

```
'<% =ControlName.ClientID %>'
```

For example:

```
DES_GetById('<% =ControlName.ClientID %>');
```

If you create the script programmatically, simply embed the **ClientID** property value. For example:

```
vScript = "DES_GetById('" + ControlName.ClientID + "');"
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Debugging Your JavaScript

Using Internet Explorer

You can debug JavaScript in Internet Explorer by using Visual Studio as your debugger. Open the **Tools**; **Internet Options** menu command and select the **Advanced** tab. Then unmark **Disable Script Debugging**.

After launching your web page from Visual Studio, switch back to Visual Studio. Then select **Debug; Windows; Script Explorer** (or **Running Documents** in VS2002/3) from the menubar. Double-click on the filename containing the JavaScript function and set a breakpoint inside the function. Now resume using your browser.

Using FireFox

Use the FireBug debugger for FireFox. Get it here: https://addons.mozilla.org/en-US/firefox/addon/1843

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Troubleshooting

Here are some issues that you may run into. Remember that technical support is available from support@PeterBlum.com. We encourage you to use this knowledge base first.

This guide contains problems specific to the **Peter's TextBoxes** module. Please see the "Troubleshooting" section of the **General Features Guide** for an extensive list of other topics including "Handling JavaScript Errors" and "Common Error Messages".

Runtime Problems

Also see the "Troubleshooting" section of the **General Features Guide.**

These controls do not filter keystrokes

There are a number of settings that can disable the client-side features of these controls.

- Browser does not support DES's code: see "Browser Support" in the **Validation User's Guide**.
- TextBoxes have a property that turns off the keyboard feature: **UseKeyboardFiltering**. Is it `false`?
- The user has shut off JavaScript on their browser.
- There is a scripting error. This requires your own custom code added to the onkeypress or onkeydown events. See the topic above, "Handling JavaScript Errors" in the Troubleshooting section of the **General Features Guide**.
- If the page is called from `Server.Transfer()`, you must add some code to the original and destination pages. See "Using Server.Transfer" in the **General Features Guide**.

Design Mode Problems

Also see the "Troubleshooting" section of the **General Features Guide.**

The TextBox control lacks some properties in the Properties Editor

The control may be the System.Web.UI.WebControl.TextBox. Change it to an Enhanced TextBox. See "Converting the ASP.NET TextBox to the Enhanced TextBox".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved