

GENERAL FEATURES OF PETER'S DATA ENTRY SUITE



Click on any of these topics to jump to them:

- ◆ [Preparing a Page for DES Controls](#)
- ◆ [Reminders As You Add Controls To The Page](#)

- ◆ [The PageManager Control.....Using](#) [Adding](#) [Properties](#)
- ◆ [NativeControlExtender ControlUsing](#) [Adding](#) [Properties](#)
- ◆ [The LocalizableLabel ControlUsing](#) [Adding](#) [Properties](#)

- ◆ [Global Settings Editor and custom.DES.config File](#)
- ◆ [Using These Controls with AJAX.....Microsoft](#) [Telerik](#) [Infragistics](#)
- ◆ [The String Lookup System.....Resources](#) [Database](#)
- ◆ [The ViewState and Preserving Properties for PostBack](#)
- ◆ [Page Level Properties and Methods Used by Most Controls](#)

- ◆ [Establishing Localization for the Web Form](#)
- ◆ [Using Style Sheets.....Add to Page](#) [Map Controls to their Files](#)
- ◆ [Using Server.Transfer](#)
- ◆ [Using Alternative HttpHandlers \(including SharePoint\)](#)
- ◆ [Expanded Properties Editor](#)
- ◆ [Browser Support](#)

- ◆ [Troubleshooting.....Handling JavaScript errors](#) [Runtime Problems](#)
- ◆ [Exploring The Current Settings: DES Debugging Reports](#)
- ◆ [Table of Contents](#)

Table of Contents

License Information.....	7
Platform Support.....	7
Technical Support and Other Assistance.....	8
Troubleshooting Sections	8
Developer's Kit.....	8
PeterBlum.Com Message Board.....	8
Getting Product Updates.....	8
Technical Support.....	8
PREPARING A PAGE FOR DES CONTROLS.....	10
REMINDERS AS YOU ADD CONTROLS TO THE PAGE.....	11
THE PAGEMANAGER CONTROL.....	12
Features	12
Using the PageManager.....	13
When to Use the PageManager.....	14
Setting Up AJAX in the PageManager Control.....	15
Adding the PageManager.....	16
PageManager Properties.....	17
AJAX Properties	18
Culture Properties	20
Draw User's Attention to Errors Properties.....	21
Validation Properties	23
Interactive Pages Properties.....	25
TextBoxes Properties.....	26
Miscellaneous Properties	27
NATIVECONTROLEXTENDER CONTROL.....	28
Features	28
Using the NativeControlExtender	29
Adding Hints.....	29
Switching to Enhanced ToolTips.....	30
Extending Buttons, LinkButtons, and ImageButtons.....	31
Validation Using the DES Validation Framework.....	31
Other Properties	31
Extending BulletedList, Menu, and TreeView	32
BulletedList.....	32
Menu	32
TreeView	32
Attach the ChangeMonitor.....	33
Provide DES Validation on AutoPostBack.....	34
Overcoming a bug in Google Chrome 1	34
Intercept the ENTER key to click a button.....	35
Extend TextBoxes to use DES's SmartChange Feature	36

Adding the NativeControlExtender	37
NativeControlExtender Properties.....	39
Control to Extend Properties.....	39
Submit the Page Properties	40
Hint Properties	43
ToolTip Properties	45
Behavior Properties	46
THE LOCALIZABLELABEL CONTROL	47
Using the LocalizableLabel Control.....	48
Adding the LocalizableLabel Control.....	49
Properties of the LocalizableLabel Control.....	50
GLOBAL SETTINGS EDITOR AND CUSTOM.DES.CONFIG FILE	51
Using the Global Settings Editor	51
Debugging the Global Settings Editor Properties	53
Programmatically Assigning Globals.....	54
Adding Globals After Config Files Load.....	54
DES.config and Custom.DES.config File	56
EXPANDED PROPERTIES EDITOR	57
Using the Expanded Properties Editor.....	58
USING THESE CONTROLS WITH AJAX	59
Using Microsoft ASP.NET AJAX.....	60
Using Telerik RadAjax	71
Using other AJAX-enabled Telerik controls	83
Using Infragistics AJAX-enabled Controls	92
Using ComponentArt CallBack	101
Using MagicAjax.....	112
Using other AJAX Products	121
AJAXManager Properties	131
Static (Shared) Properties	131
Page-Level Properties.....	131
Other AJAXManager Methods	132
PeterBlum.DES.AJAXManager.SetChildrenInAJAXUpdate.....	132
Parameters.....	132
Example	132
Analyzing the InAJAXUpdate Properties on DES Controls.....	133
Querystring parameter	133
Add a control to the page.....	133
Using the Analysis.....	133
THE STRING LOOKUP SYSTEM.....	134

Datasources	135
Elements Needed In Your DataSource.....	136
Using the Resource Manager	137
Mapping the String Group names to the associated resource files.....	138
Add .resx Files for each String Group to your Web Application.....	139
Adding These Files In Visual Studio 2005/2008	139
Adding These Files In Visual Studio 2002-2003 and Web Application Projects in VS2005/8	139
Non-Visual Studio.Net Users.....	139
Supporting Cultures	140
Entering Strings and Compiling Them	141
Using non-default Resource files and Assemblies.....	142
Properties of the PeterBlum.DES.StringLookup Class	142
Using a Database.....	146
Mapping String Group Names to their Associated Values in the Table	147
Set up a Database With The Appropriate Tables	148
Add the DESLookupString Stored Procedure	149
Provide theConnectionString to the Database	150
Writing Your Own Lookup String Event Handler.....	151
Create the Event Handler Method.....	151
Using the OnLookupString property	154
Calling The String Lookup System	155
THE VIEWSTATE AND PRESERVING PROPERTIES FOR POSTBACK.....	157
TrackProperty Method.....	157
ViewState Property – Promoting the ViewState to Public.....	157
Properties Automatically Saved in the ViewState	158
PAGE LEVEL PROPERTIES AND METHODS USED BY MOST CONTROLS.....	159
What is the PeterBlum.DES.Globals.Page property?.....	159
Properties on PeterBlum.DES.Globals.Page	160
SpinnerManager Property	163
Debugging PeterBlum.DES.Globals.Page Properties	165
Methods on PeterBlum.DES.Globals.Page.....	166
AttachCodeToEvent Method	166
ESTABLISHING LOCALIZATION FOR THE WEB FORM.....	167
Localization for the Entire Web Site	167
Default Localization for a Web Form.....	167
Change Localization Based on the User’s Culture	168
For the Entire Site	168
For the Current Page	170
USING STYLE SHEETS	171
Adding Style Sheet Files To The Page.....	172
ASP.NET 2 and above Users	172
ASP.NET 1.x Users	173

Identifying the Style Sheet File for a Specific Control..... 174

Customizing the URLs to Each Style Sheet File.....175

- Changing the URLs globally 175
 - Using the web.config file 175
 - Programmatically within Application_Start()..... 176
- Changing the URL on a Page 177
- Disabling the Output of Link Tags Globally 178
 - Using the web.config file 178
 - Programmatically within Application_Start()..... 179
- Disabling the Output of Link Tags on a Page 180

Browser Sensitive Style Sheet Class Names..... 181

- Replacing Class Names 181
 - First time – Add the CheckCssClass event 181
 - Each time – Adding a new style sheet class..... 182

Compressing and Merging Files 184

- Modifying the merging and compression features..... 184
- Troubleshooting: How to see what DES output..... 185
- Troubleshooting: Changing the URL to GetFiles.aspx..... 185
- Troubleshooting: Turning of Gzip/Deflate Compression 185

Special Parsing Features 186

Support for Your Own Style Sheet Files..... 187

- Registering your Style Sheet Files 187
 - Using web.config 187
 - Using the Application_Start() method 187
- Including your Style Sheet Files on the Page 187

USING SERVER.TRANSFER188

- Prior to the Server.Transfer Call 188
- In the Destination Page of the Server.Transfer Call 188

USING ALTERNATIVE HTTPHANDLERS (INCLUDING SHAREPOINT)188

USING A REDISTRIBUTION LICENSE189

HOW ASP.NET INFLUENCES PETER’S DATA ENTRY SUITE.....190

- Themes and Skins 190
- Automatic linking to the DES Style Sheet file 190
- Localization 190
- Validation on AutoPostBack of the TextBox and other data entry controls 190
- ValidationGroup property on submit controls 190
- Page.SetFocus vs. PeterBlum.DES.Globals.Page.InitialFocusControl..... 191
- XHTML Compatibility..... 191
- Obsolete features found in the ASP.NET 1.x assemblies of DES..... 191

BROWSER SUPPORT	192
The TrueBrowser Class.....	193
Browser Type and Version	194
Browser Capabilities.....	196
Product Feature Support	198
Customizing A TrueBrowser Object.....	200
Handling UnknownBrowsers.....	202
Extending Support For More Browsers.....	203
Adjusting the ErrorFormatter Based On The Browser	204
TROUBLESHOOTING	206
Handling JavaScript errors	207
What to do when none of the suggestions above work.....	211
Exploring The Current Settings: DES Debugging Reports	212
Running a DES Debugging Report from http://localhost	213
Running a DES Debugging Report from when the Server is not local.....	214
Access by known IP addresses	214
Access by password.....	215
Common Error Messages.....	216
What to do when you get version errors	218
Runtime Problems	219
Design Mode Problems.....	221

License Information

This document includes information for features shared amongst the modules of Peter's Data Entry Suite. All licensees have access to the controls and features described here, except where 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".

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 Sections

Most guides include 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 Message Board

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 in most guides.
- 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:

- 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.

Preparing a Page for DES Controls

While you can add Peter's Data Entry Suite ("DES") controls to your page using design mode, ASP.NET Declarative Syntax or program code, there are several items to review that make the page ideal for the controls.

- **Style Sheets** – Link to the style sheet files used by these controls. This is usually automatic in ASP.NET 2+. If using ASP.NET 1.x or they don't work in an ASP.NET 2 page, add this line to inner content of the <head> tag.

```
<%= PeterBlum.DES.StyleSheetManager.GetLinkTags() %>
```

See "Adding Style Sheet Files To The Page" for details and troubleshooting topics.

- **Register with the Page** – ASP.NET 1.x users must have this <% @Register %> tag to the top of their web form, user control or master page. *It is added automatically using the toolbox feature of Visual Studio's design mode.*

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

Note: ASP.NET 2 users should have an entry in their web.config file to avoid this. The entry was set up when running the Web Application Updater utility.

- **Convert Native Validators** – If you are adding DES's validation system to an existing page, convert that page so that the native ASP.NET validators and buttons are swapped with those from DES.

Use the **Web Application Utility** (described in the **Installation Guide**). Select the Convert Native Controls to their DES Equivalents radiobutton and One code file or web form from the DropDownList. Click Next and follow the instructions.

- **PageManager control** – If you work in design mode or ASP.NET Declarative Syntax, consider adding the PageManager control so you can easily access page-level properties. See "The PageManager Control".

(Alternatively, you can write code in Page_Load() to edit properties on **PeterBlum.DES.Globals.Page**. See "Page Level Properties and Methods Used by Most Controls".)

```
<des:PageManager id="PageManager1" runat="server" />
```

- **Global Settings** – Establish default values for features as you introduce them to your web application. Use the Global Settings Editor. See "Global Settings Editor and custom.DES.config File".
- **AJAX** – If these controls will be part of an AJAX update on this page, let DES know the AJAX framework and which controls are involved. See "Using These Controls with AJAX". *Failure to follow these directions will result in incorrect operation and JavaScript errors.*

```
<des:PageManager id="PageManager1" runat="server"
AJAXFramework="MicrosoftAJAX" />
```

- **String Lookup** – If you want these controls to get their strings from resources or a database, make sure the String Lookup System is set up. See "The String Lookup System".
- **Localization** – Many of these controls depend on culture specific information for date, time and numbers. Confirm that the page is using the desired culture settings. See "Establishing Localization for the Web Form".
- **Security** – To protect against hackers using SQL Injection and Cross Site Scripting attacks, make sure the **Peter's Input Security** module is set up (see the **Input Security Installation Guide**) and add the PageSecurityValidator control to your page.

```
<des:PageSecurityValidator id="PageSecurityValidator1" runat="server" />
```

- **Licensing** – When using a Redistribution License, add the License Key to the page. See "Using a Redistribution License".

- **Alternative HttpHandlers** – When these controls are requested by an alternative HttpHandler such as SharePoint, use this line in Page_Load():

```
PeterBlum.DES.Globals.UsingAltHttpHandler(Page)
```

See "Using Alternative HttpHandlers (including SharePoint)" for details.

Reminders As You Add Controls To The Page

- Most textboxes should have a validator attached to be sure the text is legal. Do not depend on JavaScript to be active. All the enhancements available to textboxes are not present without JavaScript.
 - Those that permit any random pattern of any characters will not have a validator, except potentially the FieldSecurityValidator from Peter's Input Security module.
 - Those that use a specific data type, like date, time, or number, should have a DataTypeCheckValidator.
 - Those that have a specific character set should have a CharacterValidator.
 - Those that have a specific pattern, like a phone number, should have a RegExValidator.
 - Those that have a limited list of valid text values should have the CompareToStringsValidator.
 - Those that have a text length limit should have the TextLengthValidator.
- When adding DES's validators, the buttons play an important role:
 - For buttons that you want to fire validation, use DES Buttons or assign the native button to a NativeControlExtender.
 - Make sure the validation group name of the button matches the group name on the validator.
 - For any button that you don't want to run validation, set its **CausesValidation** property to `false`.
- Always make sure server side validation is set up.
 - If your button's **SkipPostBackEventsWhenInvalid** property is `False`, test **PeterBlum.DES.Globals.Page.IsValid** is true inside of your button's postback event handler. If it's false, do not use the data on the page and allow the page to be redrawn.
 - If a control other than a button submits the page and you need validation, call `PeterBlum.DES.Globals.Page.Validate("groupname")` inside that control's postback event handler. Then test **PeterBlum.DES.Globals.Page.IsValid** is true before using the data.
- If you are using AJAX:
 - Set the **InAJAXUpdate** property to `true` on any DES control that is either inside of an AJAX update or references a control within the AJAX update.
 - If that control is added after the AJAX update, you may need to preregister its features when the page is initially generated. See "Using These Controls with AJAX".
- If you change the value of a property programmatically and need its value to be preserved for postback, call the `ViewStateMgr.TrackProperty()` method. See "The ViewState and Preserving Properties forPostBack".
- Use the DES TextBox control or assign a NativeControlExtender to the native TextBox to get similar functionality. *The Web Application Updater can convert native textboxes to DES textboxes.*
- Take advantage of SmartTags in the Visual Studio 2005, 2008 and Visual Web Developer design mode interface. DES controls offer their most important properties.
- DES provides extensive reporting and analysis tools to expose the details of what is going on behind the scenes. They are easy to access and make great debugging tools. See "Exploring The Current Settings".
- Instead of using the standard Properties Editor, choose the Expanded Properties Editor. Then click the  (Best Order) button to view properties in the organization recommended by PeterBlum.com. Expanded Properties Editor is available from:
 - SmartTag using "Expanded Properties Editor..."
 - Right click on the control in design mode and select the Expanded Properties Editor command.
 - The Visual Studio/VWD Properties Editor has the Expanded Properties Editor command at the bottom.

The PageManager Control

The **PeterBlum.DES.Globals.Page** object provides numerous settings that determine how the DES controls will operate. (See “What is the PeterBlum.DES.Globals.Page property?”.) Like all objects, it must be set programmatically.

The PageManager Control lets you use design mode and the ASP.NET declarative syntax to set properties of **PeterBlum.DES.Globals.Page** so you don't have to write any code. For design mode users, it makes sense to add this control to each web form using DES controls early on, so it's ready for you when you need it.

Click on any of these topics to jump to them:

- ◆ Features
- ◆ Using the PageManager
- ◆ Adding the PageManager
- ◆ PageManager Properties

Features

Each of its features relates to an aspect of DES that is covered elsewhere. The PageManager effectively groups together page-level settings in one place.

You can set these features with the PageManager control:

- Make DES aware of AJAX on the web form
- Validation page-level properties
- Culture used for localizing the page
- HintManager – rules used by the Interactive Hints feature
- SpinnerManager – rules used by spinners on textboxes
- ChangeMonitor – rules used by the ChangeMonitor
- An assortment of other properties from PeterBlum.DES.Globals.Page.

The SmartTag  for the PageManager has many useful features:

- Quickly set up AJAX
- Access to the most popular validation properties
- Run the **Global Settings Editor** (also in the controls' context menu)
- Open any of the User's Guides (also in the controls' context menu)

Note: SmartTag is a feature of design mode starting in Visual Studio 2005 and Visual Web Developer. It appears on the upper right of a control in the design mode surface.

Using the PageManager

Click on any of these topics to jump to them:

- ◆ When to Use the PageManager
- ◆ Setting Up AJAX in the PageManager Control

The PageManager control supports other DES controls and features. Here's how it is usually used:

- Add it to the page.

WARNING: *There should only be one per page. If you put it into a MasterPage it will cover all pages contained in the MasterPage. If you put it into a UserControl, make sure that UserControl is used once and no other UserControl has a PageManager.*

- If AJAX is used on the page, see "Setting Up AJAX in the PageManager Control".
- Apply any initial settings to its properties. Consider using the SmartTag  and Extended Properties Editor.
- As you work with the features of DES on the page, return and customize the settings
- Use the SmartTag  or controls' context menu when you need to open any of DES's User's Guides
- Use the SmartTag  or controls' context menu when you want to run the **Global Settings Editor**.

When to Use the PageManager

Generally you add the PageManager when you prefer to work in Visual Studio's design mode or ASP.NET Declarative Syntax. If you are comfortable assigning properties programmatically, most of the properties of PageManager can be set in your `Page_Load()` method like this:

```
PeterBlum.DES.Globals.Page.propertyname = value
```

Assigning values programmatically takes less CPU time than using the PageManager. So if you are interested in the fastest page loads possible, work programmatically.

Setting Up AJAX in the PageManager Control

When AJAX is used on the page, DES requires some additional action. See “Using These Controls with AJAX”. You can simplify the process by using the design mode features of the PageManager instead of setting the AJAXManager class programmatically.

Here's how:

1. Select your AJAX framework in the **AJAXFramework** property. *SmartTag name: “Using this AJAX Framework”*
2. Assign the correct AJAX control to the **AJAXControlID** property. *SmartTag name: “Main AJAX Control...”*

AJAX Framework	Control ID to assign to AJAXControlID
Microsoft ASP.NET AJAX	ScriptManager or ToolkitScriptManager
Telerik RadAJAX (not the “Prometheus” version)	RadAJAXManager or RadAJAXPanel
Telerik RadAJAX (using “Prometheus”)	ScriptManager
Infragistics AJAX	The “WARP” enabled control: WebAsyncRefreshPanel, UltraWebTab, UltraGauge
MagicAJAX	AjaxPanel

3. Evaluate your usage of DES controls. Are most involved in an AJAX update? If so, set **AllInAJAXUpdate** to `true`. *SmartTag name: “Most or all DES controls...”* Otherwise, set it to `false` and on individual DES controls set their **InAJAXUpdate** property to `true`.
4. In the Properties Editor, expand the **PreLoadForAJAX** property. *If in the SmartTag, first select **More Properties**.*
5. Review the list of DES features within **PreLoadForAJAX**. If any item will load for the first time after an AJAX update, set it to `true`.

Adding the PageManager



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. Add a PageManager control to the page.

Visual Studio and Visual Web Developer Users

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

Text Entry Users

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

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

Programmatically creating the PageManager control

While you can work with the PageManager control programmatically, it is a layer above features you can access directly through the **PeterBlum.DES.Globals.Page** and **PeterBlum.DES.AJAXManager** objects. Consider using their properties directly.

Guidelines for setting properties

- Design mode users can use the Properties Editor or the Expanded Properties Editor. 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"

2. Set the properties associated with the PageManager. See "PageManager Properties".

There are several properties that are objects. Their child properties are added as shown here:

- **PreLoadForAJAX** – Add to the <des:PageManager> tag like this:

```
<des:PageManager PreLoadForAJAX-PropertyName="value" />
```

- **MoreCultureInfo** – Add to the <des:PageManager> tag like this:

```
<des:PageManager MoreCultureInfo-PropertyName="value" />
```

- **HintManager** – Add to the <des:PageManager> tag like this:

```
<des:PageManager HintManager-PropertyName="value" />
```

- **ChangeMonitor** – Add to the <des:PageManager> tag like this:

```
<des:PageManager ChangeMonitor-PropertyName="value" />
```

- **SpinnerManager** – Add to the <des:PageManager> tag like this:

```
<des:PageManager SpinnerManager-PropertyName="value" />
```

PageManager Properties

Click on any of these topics to jump to them:

- ◆ [AJAX Properties](#)
- ◆ [Draw User's Attention to Errors Properties](#)
- ◆ [Validation Properties](#)
- ◆ [Interactive Pages Properties](#)
- ◆ [TextBoxes Properties](#)
- ◆ [Miscellaneous Properties](#)

AJAX Properties

These properties determine how AJAX is set up on the page. See “Setting Up AJAX in the PageManager Control”.

The Properties Editor shows these properties in the category “AJAX”.

WARNING: *Failure to set up AJAX property will result in JavaScript errors after an AJAX Update.*

- **AJAXManager** (enum PeterBlum.DES.AJAXFrameworkName) – If using AJAX, this determines which AJAX Framework that you are using.

You should also set up **AJAXControlID**, **AllInAJAXUpdate**, and **PreLoadForAJAX** properties. See “Setting Up AJAX in the PageManager Control”.

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

- None - Not using AJAX on this page, or AJAX doesn't affect these controls.
- Unknown - Do not use this.
- MicrosoftAJAX - Microsoft ASP.NET AJAX. **AJAXControlID** can be the ScriptManager, ToolkitScriptManager, or left unassigned. *When left unassigned, more CPU time is used to find the ScriptManager control.*
- TelerikRadAJAX - Telerik's RadAJAX, including the Prometheus version. For the non-Prometheus version, **AJAXControlID** must be the RadAjaxManager or RadAjaxPanel control. For the Prometheus version, **AJAXControlID** must be the ScriptManager.
- MagicAJAX - MagicAJAX. **AJAXControlID** must be the AJAXPanel control.
- InfragisticsAJAX - Infragistics AJAX enabled controls. **AJAXControlID** must be an AJAX-enabled control including the WebAsyncRefreshPanel, UltraWebTab, and UltraGauge

It defaults to `AJAXFrameworkName.None`.

If you are using any other framework, see “Using other AJAX Products”.

- **AJAXControlID** (string) – When `PeterBlum.DES.PageManager.AJAXFramework` is assigned, most frameworks need to know about a specific control that handles the callbacks. This property takes the ID to that control.

Here are the values expected based on `AJAXFramework`:

- MicrosoftAJAX - **AJAXControlID** can be the ScriptManager, ToolkitScriptManager, or left unassigned. *When left unassigned, more CPU time is used to find the ScriptManager control.*
- TelerikRadAJAX - For the non-Prometheus version, **AJAXControlID** must be the RadAjaxManager or RadAjaxPanel control. For the Prometheus version, **AJAXControlID** must be the ScriptManager.
- MagicAJAX - **AJAXControlID** must be the AJAXPanel control.
- InfragisticsAJAX - **AJAXControlID** must be an AJAX-enabled control including the WebAsyncRefreshPanel, UltraWebTab, and UltraGauge.

This control must be in the same naming container as `PageManager` or any ancestor naming container, including a `MasterPage`.

If the location of this control is not in one of those naming containers, you must programmatically set up the `AJAXManager` object as described in the “Using These Controls with AJAX”.

- **AllInAJAXUpdate** (Boolean) – When using AJAX, each DES control involved in the AJAX callback must be marked as in the AJAX update.

This property sets all that way but it's not optimal because all DES controls will transmit their data instead of just those involved in the AJAX update. For better performance, leave this `false` and set the **InAJAXUpdate** property to `true` on individual DES controls.

It defaults to `false`.

- **PreLoadForAJAX** (PeterBlum.DES.PageManager.PreLoadFeatures) – If a DES feature is not on the initial page but could be added by a callback, identify it here. This property is an object with a list of Boolean properties. They all default to false. When set to true, their associated feature is preloaded.

PageManager.PreLoadForAJAX properties	
Validators	Use when no validator is on the initial page but will be added during a callback.
ValidationSummary	Use when no ValidationSummary is on the initial page but will be added during a callback.
TextBoxes	Use when no DES TextBox is on the initial page but will be added during a callback. This includes TextBox, FilteredTextBox, IntegerTextBox, DecimalTextBox, PercentTextBox, and CurrencyTextBox. Also consider Hints, AutoPostBack.
MultiSegmentDataEntry	Use when no MultiSegmentDataEntry control is on the initial page but will be added during a callback. Also consider Hints.
Hints	Use when no hints are on the initial page but will be used during a callback. This also applies when using the hint properties on DES's textboxes and MultiSegmentDataEntry controls.
DateTextBoxes *	Use when no DateTextBox, AnniversaryTextBox, or MonthYearTextBox is on the initial page but will be added during the callback. Also consider Hints, AutoPostBack.
TimeTextBoxes *	Use when no TimeOfDayTextBox or DurationTextBox is on the initial page but will be added during the callback. Also consider Hints, AutoPostBack.
Calendar *	Use when no Calendar is on the initial page but will be added during the callback.
MultiSelectionCalendar*	Use when no MultiSelectionCalendar is on the initial page but will be added during the callback.
MonthYearPicker *	Use when no MonthYearPicker or PopupMonthYearPicker is on the initial page but will be added during the callback.
TimePicker *	Use when no TimePicker or PopupTimePicker is on the initial page but will be added during the callback.
SpecialDates	Use when no SpecialDates control is on the initial page but will be added during the callback.
ContextMenu	Use when no ContextMenu control is on the initial page but will be added during the callback.
FieldStateControllers	Use when no type of FieldStateController is on the initial page but will be added during a callback. This includes FieldStateController, MultiFieldStateController, FSCOnCommand, and MultiFSCOnCommand.
CalculationControllers	Use when no CalculationController is on the initial page but will be added during a callback.
TextCounter	Use when no TextCounter is on the initial page but will be added during the callback.
AutoPostBack	Use when you validate on AutoPostBack.
ChangeMonitor	Use when the ChangeMonitor is added to the page after postback.
SubmitControls	Use when no Button, LinkButton, or ImageButton is on the initial page but will be added during the callback.
DisableOnSubmit	Use when no Button uses the DisableOnSubmit feature on the initial page but will be used during a callback.
Popups	Use when you create a control using the Popup code.

* Also requires calling the PreloadForAJAX() method in Page_Load()

ASP.NET Declarative Syntax

```
<des:PageManager PreLoadForAJAX-PropertyName="value" />
```

Culture Properties

DES is localized through these two properties: **PeterBlum.DES.Globals.Page.CultureInfo** and **PeterBlum.DES.Globals.Page.MoreCultureInfo**. The PageManager properties allow changing some of the most common aspects of these objects.

The Properties Editor shows these properties under the category "Culture".

- **CultureName** (string) – Overrides the page's current CultureInfo using the culture name specified.

The name must be compatible with `CultureInfo.CreateSpecificCulture()`, such as 'en-US'. See this topic for a list of valid Culture Names: <http://msdn2.microsoft.com/en-us/library/system.globalization.cultureinfo.aspx>.

When assigned, it will set **PeterBlum.DES.Globals.Page.CultureInfo** with a CultureInfo object defined by this culture name.

When blank, **PeterBlum.DES.Globals.Page.CultureInfo** uses the page's or site's culture as described in "Establishing Localization for the Web Form".

It defaults to "".

- **MoreCultureInfo** (ClientSideCultureInfo) – Additional culture related properties that do not exist in the CultureInfo class. They are only used by the **Peter's Date and Time** module.

It defines the CenturyBreak, additional date separators for the DateTextBox, the first day of the week, and time format for the TimeOfDayTextBox. `PeterBlum.DES.ClientSideCultureInfo` is described in the "The PeterBlum.DES.Globals.Page.MoreCultureInfo property" of the **Date and Time User's Guide**.

ASP.NET Declarative Syntax

```
<des:PageManager MoreCultureInfo-PropertyName="value" />
```

- **ShortDatePatternOverride** (string) – When assigned, it replaces the value of **PeterBlum.DES.Globals.Page.CultureInfo.DateTimeFormat.ShortDatePattern**.

Always use the standard ShortDatePattern rules of one or two 'M' characters for month, 4 'y' characters for year, and one or two 'd' characters for day. Always include a single character date separator to delimit it. For example: yyyy-MM-d, dd.yyyy.MM, d/M/yyyy.

The date separator character will be extracted from your pattern automatically and assigned to **PeterBlum.DES.Globals.Page.CultureInfo.DateTimeFormat.DateSeparator**.

It defaults to "".

- **LongDatePatternOverride** (string) – When assigned, it replaces the value of **PeterBlum.DES.Globals.Page.CultureInfo.DateTimeFormat.LongDatePattern**.

Always use the standard ShortDatePattern rules of 3 or 4 'M' characters for month, 4 'y' characters for year, and one or two 'd' characters for day. Use any formatting characters between them. For example: MMMM dd, yyyy.

It defaults to "".

Draw User's Attention to Errors Properties

These properties are used by the DES Validation Framework and are described in the **Validation User's Guide** under the topic "Drawing User's Attention to the Error". Each of these properties has a default value which will use a global setting defined in the **Global Settings Editor**.

The Properties Editor shows these properties under the category "Draw Users Attention to Errors".

Alert: These properties are only available when you have a license covering Peter's Professional Validation.

- **FocusOnChange** (enum TrueFalseDefault) – Set focus to the control which caused a validation error immediately after editing and focus leaves the control. When set to `TrueFalseDefault.Default`, the page uses this global setting: **DefaultFocusOnChange**.
- **FocusOnSubmit** (enum TrueFalseDefault) – Set focus to the control when the user attempts to submit the page and a validation error is found. When set to `TrueFalseDefault.Default`, the page uses this global setting: **DefaultFocusOnSubmit**.
- **ShowAlertOnChange** (enum TrueFalseDefault) – Determines if an alert appears when a validation error occurs immediately after an edit. When set to `TrueFalseDefault.Default`, the page uses this global setting: **DefaultShowAlertOnChange**.
- **ShowAlertOnSubmit** (enum TrueFalseDefault) – Determines if an alert appears when the user attempts to submit the page and a validation error is found. When set to `TrueFalseDefault.Default`, the page uses this global setting: **DefaultShowAlertOnSubmit**.
- **ChangeStyleOnControlsWithError** (enum TrueFalseDefault) – Determines if controls with a validation error change their style, using style sheets, to show an error.

By default, the style sheet classes `DESVALFieldWithError`, `DESVALListWithError`, and `DESVALCheckBoxWithError` are where you set their appearance in the

DES\Appearance\Validation\Validation.css file. These style sheet classes are merged with the original class defined on the `CssClass` property of the control.

When set to `TrueFalseDefault.Default`, the page uses this global setting:

DefaultChangeStyleOnControlsWithError.

- **HiliteFieldsNearbyError** (enum TrueFalseDefault) – Determines if the `HiliteFields` feature is enabled. When it is, fields identified by the validator's `Labels` and **HiliteFields** property will change their appearance when that validator detects an error.

By default, the style sheet classes `DESVALTextHiliteFields` and `DESVALNonTextHiliteFields` are where you set their appearance in the **DES\Appearance\Validation\Validation.css** file. These style sheet classes are merged with the original class defined on the `CssClass` property of the control.

When set to `TrueFalseDefault.Default`, the page uses this global setting: **DefaultHiliteFieldsNearbyError**.

- **BlinkOnChange** (enum PeterBlum.DES.BlinkMode) – Enables blinking and determines how many blinks occur to a validator error message after a change to the field.

When set to `BlinkMode.Off`, the page uses this global setting: **DefaultBlinkOnChange**.

- **BlinkOnSubmit** (enum PeterBlum.DES.BlinkMode) – Enables blinking and determines how many blinks occur to a validator error message when the user submits the page and there are validation errors.

When set to `BlinkMode.Off`, the page uses this global setting: **DefaultBlinkOnSubmit**.

- **BlinkTime** (integer) – The number of milliseconds between blinks of an error message. Used by **BlinkOnChange** and **BlinkOnSubmit** properties.

When set to 0, the page uses this global setting: **DefaultBlinkTime**.

- **AlertTemplate** (string) – Places text before and after the validation error messages when they are shown in an alert.
Used by the **ShowAlertOnChange** and **ShowAlertOnSubmit** features.
The token “{0}” will be replaced by the error messages. For example, “Please correct these errors:\n{0}.”.
When “”, the page uses this global setting: **DefaultAlertTemplate**.
- **AlertTemplateLookupID** (string) – An alternative to **AlertTemplate** that retrieves a string from the String Lookup System. Strings must be in the String Group of ValidationMisc.
When “”, the page uses this global setting: **DefaultAlertTemplateLookupID**.
- **AlertErrorLeadText** (string) – Places text before each error message shown in an alert. Use it to denote a new message. For example, “-” or “*”.
When “”, the page uses this global setting: **DefaultAlertErrorLeadText**.
- **AlertErrorListStyle** (enum TrueFalseDefault) – Used when showing error messages in an alert. Formats the list of error messages shown in an alert.
When `TrueFalseDefault.True`, error messages are listed on separate lines.
When `TrueFalseDefault.False`, error messages are listed in a single paragraph style.
When `TrueFalseDefault.Default`, the page uses this global setting: **DefaultAlertErrorListStyle**.
- **FocusAfterAlert** (enum TrueFalseDefault) – Set focus to the control which caused a validation error when the user clicks on an `ErrorFormatter` that opens an alert. The focus is set after the alert is closed.
When set to `TrueFalseDefault.Default`, the page uses this global setting: **DefaultFocusAfterAlert**.

Validation Properties

Here are validation properties in addition to those that draw the user's attention to errors (see above). These properties are used by the DES Validation Framework and are described in the **Validation User's Guide**. Each of these properties has a default value which will use a global setting defined in the **Global Settings Editor**.

The Properties Editor shows these properties under the category "Validation".

Alert: These properties are only available when you have a license covering Peter's Professional Validation.

- **ConfirmMessage** (string) – When assigned, this is used as the confirm message shown by validation when the page is submitted. When unassigned, the page uses the global setting: **DefaultConfirmMessage**.
Alert: Requires licenses covering Peter's Professional Validation and Peter's Interactive Pages.
- **ConfirmMessageLookupID** (string) – An alternative to **ConfirmMessage** that retrieves a string from the String Lookup System. Strings must be in the Confirm String Group.
When "", the page uses this global setting: **DefaultConfirmMessageLookupID**.
Alert: Requires licenses covering Peter's Professional Validation and Peter's Interactive Pages.
- **ConfirmMessageGroup** (string) - The validation group name that must match the submit button's group before showing the confirm message box.
When "", the page uses this global setting: **DefaultConfirmMessageGroup**.
Alert: Requires licenses covering Peter's Professional Validation and Peter's Interactive Pages.
- **DefaultGroup** (string) – The validation group name that is used when the user hits ENTER on the page to submit. Not used when blank.
In this situation, the buttons don't get their client-side onclick event handler run. So no button applies its **Group** property to the submission code. So this is a fall-back.
It actually only establishes an initial group on the page. Once the user clicks a button, it changes the default to that group. It defaults to "" and does not have a global setting.
- **AutoDisableValidators** (enum TrueFalseDefault) – Determines if validators evaluate controls that are hidden or disabled without using the **Enabler** property with a **VisibleCondition** and **EnabledCondition**.
When `TrueFalseDefault.True`, validators will not attempt to evaluate controls that are hidden or disabled.
When `TrueFalseDefault.False`, they will unless you establish the **Enabler** property with a **VisibleCondition** and **EnabledCondition**.
When set to `TrueFalseDefault.Default`, the page uses this global setting: **DefaultAutoDisableValidators**.
- **SubmitOrder** (enum SubmitOrderType) – Determines the order of these three actions that occur when the page is submitted:
 - Validate the page.
 - Show the Confirm messagebox when a **ConfirmMessage** property is used, either here in the **PageManager** or on a DES button.
 - Run the Custom Submit Function when the **CustomSubmitFunctionName** property is used.
 When set to `SubmitOrderType.ConfirmCustomValidate`, the page uses this global setting: **DefaultSubmitOrder**.

- **CustomSubmitFunctionName** (string) – The name of a client-side function that is called when the page is submitted for validation. It allows you to extend the submission logic.

The function must take one parameter, the group name, which is a string. It must return a Boolean value where true means continue and false means stop. See this property in the **Validation User's Guide** for an example.

The function is run amongst three actions: validation, confirm message, and custom submit function in the order determined by **SubmitOrder**.

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

It defaults to "". There is no global setting for this property.

***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.

- **PostValidationUpdateScript** (string) – JavaScript code that will be executed each time validation occurs on the client-side. It will run after validation is applied. One use is to relocate absolute positioned elements after validator error messages have caused the page to reposition non-absolute position elements.

You can enter any JavaScript statements you want into this string. Your string will be executed by using the JavaScript `eval()` function. This function may be called even if nothing visually changed on the page.

When "", this feature is not used.

It defaults to "". There is no global setting for this property.

Interactive Pages Properties

These properties are features of the **Peter's Interactive Pages** module.

The Properties Editor shows these properties under the category "Interactive Pages".

Alert: These properties are only available when you have a license covering Peter's Interactive Pages.

- **HintManager** (HintManager) – The `PeterBlum.DES.HintManager` class is used by the Interactive Hints system to define the formatting and behavior of hints and tooltips. See "Interactive Hints" in the **Interactive Pages User's Guide**.

Use its properties to:

- Define shared HintFormatters. Each defines how a hint appears on the page. They have a name which is assigned to individual controls in their **SharedHintFormatterName** property.
- Determine if validator errors are merged into the hints.
- Switch from standard HTML tooltips to DES's PopupViews.

ASP.NET Declarative Syntax

```
<des:PageManager HintManager-PropertyName="value" />
```

- **ChangeMonitor** (ChangeMonitor) – The `PeterBlum.DES.ChangeMonitor` class is used by the Change Monitor system to enable it and define its behavior. See "Change Monitor" in the **Interactive Pages User's Guide**.

ASP.NET Declarative Syntax

```
<des:PageManager ChangeMonitor-PropertyName="value" />
```

TextBoxes Properties

These properties are used by textboxes in **Peter's TextBoxes** and **Peter's Date and Time** modules.

The Properties Editor shows these properties under the category "Interactive Pages".

Alert: These properties are only available when you have a license covering Peter's TextBoxes or Peter's Date and Time.

- **SpinnerManager** (SpinnerManager) – The `PeterBlum.DES.SpinnerManager` class is used by textboxes that offer spinners including `TimeOfDayTextBox`, `DurationTextBox`, and all numeric textboxes of **Peter's TextBoxes**. See "SpinnerManager Property".

ASP.NET Declarative Syntax

```
<des:PageManager SpinnerManager-PropertyName="value" />
```

- **ValueWhenBlankMode** (enum `ValueWhenBlankMode`) – Used by TextBoxes to determine how setting and removing the focus updates the textbox if its value is considered blank. It may restore the text value to "" and change the style sheet class to its original value.

Use the **ValueWhenBlank** property on TextBoxes to establish the text for when it is blank.

When set to `ValueWhenBlankMode.RemoveBoth`, the page uses this global setting:

DefaultValueWhenBlankMode.

Miscellaneous Properties

The Properties Editor shows these properties under the category “Misc”.

- **EnableButtonImageEffects** (enum EnableButtonImageEffects) – Many buttons can show up to 3 images: normal, pressed, and mouseover. You only need to specify the name of the normal image and provide two more with the same name + “pressed” and “mouseover”. DES will “sniff” your local folder to detect these files. It’s sniffing cannot see all possible URLs, including those starting with “http://”. Use this property to override the sniffer. See the **PeterBlum.DES.Globals.Page.EnableButtonEffects** property.
- **SetFocusFunctionName** (string) – The name of a client-side function that is called when DES sets focus to a field. It allows you to modify or replace the set focus logic, particularly to show an invisible control. See the **PeterBlum.DES.Globals.Page.SetFocusFunctionName** property.
- **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 forPostBack”.

NativeControlExtender Control

While DES provides enhanced versions of the TextBox, Button, LinkButton, and ImageButton, you may want to continue using the native ASP.NET versions of these controls on your page. The NativeControlExtender lets you add some DES features to the native controls. It also extends controls for which DES has no equivalent.

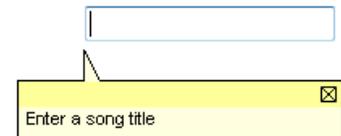
Note: Switching from native to DES versions of controls is very easy and is probably worth doing, so you don't have two controls instead of one, for each button and textbox. Run the Web Application Updater to Convert Native controls on any page or throughout your web application. See the Installation Guide.

Click on any of these topics to jump to them:

- ◆ Features
- ◆ Using the NativeControlExtender
 - Adding Hints
 - Switching to Enhanced ToolTips
 - Extending Buttons, LinkButtons, and ImageButtons
 - Extending BulletedList, Menu, and TreeView
 - Attach the ChangeMonitor
 - Provide DES Validation on AutoPostBack
 - Intercept the ENTER key to click a button
 - Extend TextBoxes to use DES's SmartChange Feature
- ◆ Adding the NativeControlExtender
- ◆ NativeControlExtender Properties

Features

- Apply the Interactive Hints system to almost any control. For controls that allow focus, show a Hint on a PopUpView or Label. For most, switch from the standard tooltip to a PopUpView (shown to the right).
- Extend Buttons, LinkButtons, and ImageButtons, with these DES features:
 - DES Validation
 - Disable On Submit
 - ConfirmMessage
- Extend these controls which can submit the page to offer client-side DES validation and ConfirmMessages:
 - BulletedList when **DisplayMode** = LinkButton.
 - Menu for selected menu items
 - TreeView for TreeNodes with a **SelectAction** of Select or SelectExpand. *ConfirmMessages are not offered on this control.*
- ChangeMonitor monitors edits on controls that support client-side onchange and onclick events.
- Provide DES validation on AutoPostBack.
- Intercept the ENTER key and use it to click a button.
- Extend TextBoxes with DES's SmartChange feature that fires the client-side onchange event in cases where it would be expected but doesn't happen: after using the AutoComplete menu and after a programmatic edit.



Using the NativeControlExtender

It's easy to use the NativeControlExtender. Assign the **ControlIDToExtend** property to the ID of the control that you need to extend. Then set the desired properties.

Click on any of these topics to jump to them:

- ◆ Adding Hints
- ◆ Switching to Enhanced ToolTips
- ◆ Extending Buttons, LinkButtons, and ImageButtons
- ◆ Extending BulletedList, Menu, and TreeView
- ◆ Attach the ChangeMonitor
- ◆ Provide DES Validation on AutoPostBack
- ◆ Intercept the ENTER key to click a button
- ◆ Extend TextBoxes to use DES's SmartChange Feature

Adding Hints

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.

Use hints with controls that support the client-side onfocus and onblur events. This includes these HTML form tags: <input>, <select>, and <textarea>. Their ASP.NET control equivalents are TextBoxes, CheckBoxes, RadioButtons, DropDownLists, and ListBoxes. It may work with some third party controls too (you will have to try it to see).

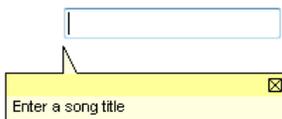
See the "Interactive Hints" section of the **Interactive Pages User's Guide** for details.

Assign your hint text to the **Hint** property. If you are using 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.ToolTipAsHints** property to **False**.

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. **HintHelp** can be used for other purposes, as determined by the **PopupView.HelpBehavior** property. See the "Defining PopupViews" section of the **Interactive Pages User's Guide**.

The format of hints is determined by a **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**. 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

Switching to Enhanced ToolTips

When you have a license for the Interactive Pages module, you can replace the browser's ToolTips with DES's Enhanced ToolTips, as described in the "Enhanced ToolTips" section of the **Interactive Pages User's Guide**.

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.

When you set **PeterBlum.DES.Globals.Page.HintManager.EnableToolTipsUsePopupViews** or **PageManager.HintManager.EnableToolTipsUsePopupViews** to **True**, any of DES's controls that have a ToolTip or Hint defined will show a ToolTip with a PopupView. This can be applied to non-DES controls too.

Just assign a **NativeControlExtender** to the control and use the **ToolTip** property either on your non-DES control or the **NativeControlExtender**. If you are already using the **Hint** property, it can be used as the text of a ToolTip unless you want its text to be different.

The **PopupView**'s appearance is defined with settings established in the "PopupView definitions used by HintFormatters" section of the **Global Settings Editor**. In the **Global Settings Editor**, you define a **PopupView** and give it a name. *There are several predefined PopupViews with different colors and widths.* See "Defining PopupViews" in the **Interactive Pages User's Guide**.

You can establish the **PopupView** used on a **ToolTip** by specifying its name in the **ToolTipUsesPopupViewName** property of the **NativeControlExtender**. If it is set to "{DEFAULT}", it uses the value from the **DefaultToolTipPopupViewName** property of the "HintManager Defaults" topic of the **Global Setting Editor**.

The screenshot shows two panels from the Global Settings Editor. The left panel, titled "PopupView definitions used by HintFormatters", lists various predefined popup views such as "LtRed-Small", "Callout-Medium", and "ToolTip-Small". The right panel, titled "HintManager Defaults", contains a table of settings. The "DefaultToolTipPopupViewName" is highlighted in blue and set to "ToolTip-Small". Below this table is a section for "DefaultToolTipPopupViewName" with a description: "The default PopupView when tooltips are replaced by PopupView (HintManager.EnableToolTipsUsePopupViews is true)."

HintManager Defaults	
DefaultHintPopupViewBodyImageUrl	
DefaultHintPopupViewName	LtYellow-Small
DefaultHintsShowErrors	Hint
DefaultHintsShowErrorsCssClass	
DefaultHintsShowErrorsCssClass2	
DefaultHintsShowErrorsSeparator	
DefaultHintsShowTextCounterSeparator	
DefaultToolTipsAsHints	True
ToolTips Replaced by PopupViews	
DefaultEnableToolTipsUsePopupViews	False
DefaultToolTipPopupViewName	ToolTip-Small
ToolTipHideDelay	300
ToolTipHideOnClick	False
ToolTipHideOnTyping	False
ToolTipShowDelay	500

DefaultToolTipPopupViewName
The default PopupView when tooltips are replaced by PopupView (HintManager.EnableToolTipsUsePopupViews is true). Ho

Extending Buttons, LinkButtons, and ImageButtons

DES's Buttons have several extensions: support for DES Validation, disable on submit, enabled by the ChangeMonitor, showing a confirm message, and handling a click when the button moves from under the mouse. While it is easier to just switch a page from the native buttons to DES buttons (use the Web Application Updater), the same features are found in the NativeControlExtender.

Validation Using the DES Validation Framework

Simply by attaching the NativeControlExtender, your button is set up for client-side validation using the DES Validation Framework. The NativeControlExtender will respect your button's **CausesValidation** property and not set up validation if it is `false`. It will also use the **ValidationGroup** property from your button. Alternatively, you can set up a validation group name in the NativeControlExtender's **Group** property.

Unlike with the Native Validation Framework, you can use the group name "*" to validate ALL groups. When the button is shown on multiple rows (naming containers) of a GridView or Repeater, you can make each row have a unique group name by adding a plus (+) character as the first character of the group name.

Server Side Validation

You **must** still set up server side validation on any submit control that should validate. Your button's **Click** or **Command** event handler method should look like this:

[C#]

```
protected void ControlName_Click(object sender, System.EventArgs e)
{
    PeterBlum.DES.Globals.Page.Validate("validation group name");
    if (PeterBlum.DES.Globals.Page.IsValid)
    {
        // code to save or use the data
    }
}
```

[VB]

```
Protected Sub ControlName_Click(ByVal sender As object, ByVal e As System.EventArgs)
    PeterBlum.DES.Globals.Page.Validate("validation group name")
    If PeterBlum.DES.Globals.Page.IsValid Then
        ' code to save or use the data
    End If
End Sub
```

Other Properties

- **ConfirmMessage** is the text of a confirmation message shown when the button is pressed.
- **DisableOnSubmit** will disable the button after it is clicked to reduce the chance of extra page submissions.
- **MayMoveOnClick** resolves an issue where the button may jump just as the user attempts to click it, requiring a second click. This often happens after editing a field that has a validator error and the user immediately clicks on the button.

Extending BulletedList, Menu, and TreeView

The BulletedList, Menu, and TreeView controls all can post back. If you want them to run client-side validation and optionally display a confirmation message prior to posting back, use a NativeControlExtender.

Click on any of these topics to jump to them:

- ◆ BulletedList
- ◆ Menu
- ◆ TreeView

BulletedList

The BulletedList can submit the page when its **DisplayMode** property is set to `LinkButton`. The `NativeControlExtender` will use the **CausesValidation** and **ValidationGroup** properties already on the BulletedList to establish client-side validation. You must still set up server side validation in its postback event handler method. See “Server Side Validation”.

Set the **ConfirmMessage** property if you want a confirmation message prior to submitting the page.

Menu

Only applies to the System.Web.UI.WebControls.Menu

Menus can post back. Client-side validation may be appropriate for some commands. You mark those commands by assigning the “{SUBMIT}” token into their **NavigateUrl** property.

The “{SUBMIT}” token takes additional parameters to define the validation group and elect to use the **ConfirmMessage**. The parameters are in this colon delimited format:

```
{SUBMIT:group=[validationgroupname]:confirm}
```

When `:group=` is defined, validation is used with the validation group name specified. It supports a single group name or “*”. For the blank group name, just use `group=`.

When `:confirm` is used, show the confirm message supplied to this function. If validation is also used, the confirm message will appear in the order determined by `PeterBlum.DES.Globals.Page.SubmitOrder` or `PageManager.SubmitOrder`.

```
<asp:Menu ID="Menu1" runat="server" OnMenuItemClick="Menu1_MenuItemClick">
  <Items>
    <asp:MenuItem Text="Confirm and validate" Value="Confirm and validate"
      NavigateUrl="{SUBMIT:group=:confirm}"></asp:MenuItem>
    <asp:MenuItem Text="Confirm" Value="Confirm"
      NavigateUrl="{SUBMIT:confirm}"></asp:MenuItem>
    <asp:MenuItem Text="New Item3" Value="New Item3">
      <asp:MenuItem Text="Validate group 1" Value="New Item 3.1"
        NavigateUrl="{SUBMIT:group=group1}"></asp:MenuItem>
      <asp:MenuItem Text="Validate group 1 and confirm" Value="New Item 3.2"
        NavigateUrl="{SUBMIT:group=group1:confirm}"></asp:MenuItem>
      <asp:MenuItem Text="Nothing" Value="Nothing" ></asp:MenuItem>
    </asp:MenuItem>
  </Items>
</asp:Menu>
```

You must still set up server side validation in its postback event handler method. See “Server Side Validation”.

TreeView

Only applies to the System.Web.UI.WebControls.TreeView

The TreeView can provide client-side validation (but no confirm message) on any `TreeNode` with a **SelectAction** of `Select` or `SelectExpand`. It uses the validation group name defined in the **Group** property.

You must still set up server side validation in its postback event handler method. See “Server Side Validation”.

Attach the ChangeMonitor

The ChangeMonitor watches for edits on the page and changes the enabled state of buttons after an edit is detected. See “ChangeMonitor” in the **Interactive Pages User’s Guide**.

When the ChangeMonitor is enabled, data entry controls assigned to the NativeControlExtender will notify the ChangeMonitor when they are changed. *This supports the native controls and those third party controls defined in the <ThirdPartyControl> section of the custom.des.config file as described in the **Installation Guide**.*

Provide DES Validation on AutoPostBack

Many controls offer the **AutoPostBack** property to post back when they are changed. Sometimes you want to provide client-side validation and block the postback when there is an error. You can use the `NativeControlExtender` to not only install support for DES's Validation Framework, but also to preserve the focus position after postback is completed.

Set the non-DES control's **AutoPostBack** property to `true` and attach the `NativeControlExtender`.

Determine what you want to validate with the **AutoPostBackValidates** property:

- When set to `Control`, it validates all validators attached to your control.
- When set to `ValidationGroup`, it validates based on the validation group supplied. Define the validation group name in either the control's **ValidationGroup** property or `NativeControlExtender`'s **Group** property. It supports the group name "*" to evaluate all validation groups and the "+" character in front of the group name for controls repeated in naming containers.

If you want to preserve focus, set **AutoPostBackTracksFocus** to `true`. *Not available in ASP.NET 1.x.* You can track focus even when **AutoPostBackValidates** is set to `No`.

Overcoming a bug in Google Chrome 1

When a `DropDownList` has **AutoPostBack** set to `true` and you assign a DES validator to that `DropDownList`, Google Chrome v1 will generate a JavaScript error. Use the `NativeControlExtender` to avoid this. Add it to any `DropDownList` whose **AutoPostBack** property is set to `true` and has validators.

Internally the `NativeControlExtender` swaps the original `AutoPostBack` scripts with alternatives that work with all browsers.

Intercept the ENTER key to click a button

If you want to direct the ENTER key to click a button, assign a control where typing will occur and set the **EnterSubmitsControlID** property to the button's ID. You can select an actual data entry control or a containing control, like a Panel. The containing control will capture the ENTER key for all controls it contains. *The contain control must generate an HTML tag that supports the onkeypress event.*

Extend TextBoxes to use DES's SmartChange Feature

DES's SmartChange feature assures you that a textbox will fire its client-side onchange event even when the browser normally would not. The onchange event is very important in data entry. It updates validators, notifies the ChangeMonitor, and tells the control to reformat its contents.

The browser will not fire this event when you pick from the AutoComplete list to edit the control. If you write JavaScript to change the control while focus is in the control, that change will not be detected either. Both are addressed by the SmartChange feature.

To use it, simply attach the NativeControlExtender to your textbox. There are no additional settings.

Adding the NativeControlExtender



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”. It covers issues like style sheets, AJAX, and localization.
2. Add a NativeControlExtender control to the page. Since it does not add any HTML to the page, it can be anywhere.

Visual Studio and Visual Web Developer Users

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

Text Entry Users

Add the control:

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

Programmatically creating the NativeControlExtender control

- Identify the control which you will add the NativeControlExtender control to its **Controls** collection. Like all ASP.NET controls, the NativeControlExtender 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 NativeControlExtender control class. The constructor takes no parameters.
- Assign the **ID** property.
- Add the NativeControlExtender control to the **Controls** collection.

In this example, the NativeControlExtender is created with an **ID** of “NativeControlExtender1”. It is added to Placeholder1.

[C#]

```
PeterBlum.DES.NativeControlExtender vNativeControlExtender =
    new PeterBlum.DES.NativeControlExtender();
vNativeControlExtender.ID = "NativeControlExtender1";
Placeholder1.Controls.Add(vNativeControlExtender);
```

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

[VB]

```
Dim vNativeControlExtender As PeterBlum.DES.NativeControlExtender = _
    New PeterBlum.DES.NativeControlExtender()
vNativeControlExtender.ID = "NativeControlExtender1"
Placeholder1.Controls.Add(vNativeControlExtender)
```

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

3. Assign the **ControlIDToExtend** property to the ID of the control you want to extend. If that control is not in the same or a parent naming container, set it programmatically with the **ControlToExtend** property.

Guidelines for setting properties

- Design mode users can use the Properties Editor or the Expanded Properties Editor. 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"`

4. Set other properties to extend your control. See “NativeControlExtender Properties”.

Notable Properties

The “Using the NativeControlExtender” section provides details on these properties.

- ◆ Adding Hints
- ◆ Switching to Enhanced ToolTips
- ◆ Extending Buttons, LinkButtons, and ImageButtons
- ◆ Extending BulletedList, Menu, and TreeView
- ◆ Attach the ChangeMonitor
- ◆ Provide DES Validation on AutoPostBack
- ◆ Intercept the ENTER key to click a button
- ◆ Extend TextBoxes to use DES’s SmartChange Feature

5. Here are some other considerations:

- If you are using an AJAX system to update this control, set the **InAJAXUpdate** property to `true`. See “Using These Controls with AJAX”.
- 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”.
- If you encounter errors, see the “Troubleshooting” section for extensive topics based on several years of tech support’s experience with customers.

NativeControlExtender Properties

The NativeControlExtender is subclassed from System.Web.UI.Control. It adds no HTML of its own to the page.

Click on any of these topics to jump to them:

- ◆ Control to Extend Properties
- ◆ Submit the Page Properties
- ◆ Hint Properties
- ◆ ToolTip Properties
- ◆ Behavior Properties

Control to Extend Properties

The Properties Editor shows these properties in the “Control To Extend” category.

- **ControlIDToExtend** (string) – The ID to the control to be extended. Either it or ControlToExtend must be assigned. Use ControlToExtend when the control is not in the same or a parent naming container with the NativeControlExtender. It accepts the ID of nearly any native ASP.NET control that generates HTML. See “Using the NativeControlExtender” for recommendations on the controls to use with the NativeControlExtender.
- **ControlToExtend** (Control) – A reference to the control to be extended. It is an alternative to **ControlIDToExtend** that allows the control to be anywhere on the page instead of the same naming container as the Calendar control. You must assign it programmatically.

When assigned, it overrides the value of **ControlIDToExtend**.

Submit the Page Properties

The Properties Editor shows these properties in the “Submit The Page” category.

- **Group** (string) – Determines which validators are invoked when the Control To Extend control attempts to submit the page. *Only supported when using the DES Validation Framework.*

When Control To Extend already has its own **ValidationGroup** property, such as a Button, the value of **ValidationGroup** will be used when this property is blank. When **Group** is assigned, it always overrides the **ValidationGroup** property.

Do not use this with the ChangeMonitor. Set up ChangeMonitor group names in the **ChangeMonitorGroups** property.

The validators whose **Group** property matches this value will be evaluated.

Group names are blank by default. When left blank, this runs all validators whose **Group** property is "".

You can also use the string “*” to run every validator on the page.

When the button is shown on multiple rows (naming containers) of a GridView or Repeater, you can make each row have a unique group name by adding a plus (+) character as the first character of the group name. Just be sure to use an identical name in the validators associated with this button.

It defaults to "".

- **AutoPostBackValidates** (enum PeterBlum.DES.AutoPostBackValidates) – When Control To Extend has its **AutoPostBack** property set to `true`, this determines if autopostback will first run client-side validation before posting back. If there is validation error, it does not post back. *Only supported when using the DES Validation Framework.*

Requirements: The Control To Extend must have an **AutoPostBack** property for this feature to be applied.

DES can either validate the control itself or all validators in the validation group defined by the **Group** property.

It does not provide any server side validation. The idea is to avoid a round trip when the data entered is meaningless. Yet, you should act defensively to protect against hacking attempts by using the PageSecurityValidator or otherwise detecting illegal data passed by the controls on the page so it cannot be used in SQL statements.

If client-side validation is not set up, **AutoPostBack** does its normal processing without client-side validation.

In ASP.NET 2.0 and higher, the Control To Extend may have a **CausesValidation** property. It is **not** used. The value `AutoPostBackValidates.ValidationGroup` is its replacement.

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

- `No` - AutoPostBack does not validate. Postback always occurs.
- `Control` - AutoPostBack runs all validators associated with the Control To Extend.
- `ValidationGroup` - AutoPostBack runs all validators associated with the validation group. The validation group name is specified in the **Group** property.

It defaults to `AutoPostBackValidates.Control`.

Alert: This property requires a license that covers the Peter's Professional Validation module.

- **AutoPostBackTracksFocus** (Boolean) – *In ASP.NET 2.0 and higher.* When Control To Extend has its **AutoPostBack** property set to `true`, this allows some browsers to set the focus to the control that last had focus. The control may either be the Control To Extend or the control the user has moved the focus to which invoked autopostback.

Requirements: The Control To Extend must have an **AutoPostBack** property for this feature to be applied.

When `true`, it restores focus back to this control after auto post back completes.

When `false`, it does not affect focus.

It defaults to `false`.

Alert: This property requires a license that covers the Peter's Professional Validation module.

- **ConfirmMessage** (string) – Provides a confirmation message to controls that submit the page, including Button, LinkButton, ImageButton, BulletedList (when **DisplayMode** = `LinkButton`), and Menu (for MenuItems with `NavigateUrl = "{SUBMIT:confirm}"`).

When using the DES Validation Framework and the Control to Extend runs the validators (its **CausesValidation** property is true), there is a default confirmation message that is used when this property is blank. The default is **PeterBlum.DES.Globals.Page.ConfirmMessage** or **PageManager.ConfirmMessage**.

A confirmation message uses the JavaScript `confirm()` function to display the text of this property. It offers a **Yes** and **No** button. When **No** is clicked, the control will not post back.

It defaults to "".

Alert: This property requires a license that covers the Peter's Interactive Pages module.

- **ConfirmMessageLookupID** (string) – Gets the value for **ConfirmMessage** through the String Lookup System. (See “The String Lookup System”.) The LookupID and its value should be defined within the String Group of Confirm. If no match is found OR this is blank, **ConfirmMessage** 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 "".

Alert: This property requires a license that covers the Peter's Interactive Pages module.

- **DisableOnSubmit** (Boolean) – When `true` and Control To Extend is a Button, LinkButton, or ImageButton, the control will be disabled after the page submits.

When `true`, the control will disable on submit. If the button is set up for AJAX, it will be re-enabled when the AJAX callback completes. If using an ImageButton, DES will apply an opaque style to give the image a dimmed effect.

It defaults to `false`.

Alert: This property requires a license that covers the Peter's Interactive Pages module.

- **MayMoveOnClick** (Boolean) – When the Control To Extend is a Button, LinkButton, or ImageButton and it requires an extra click to submit the page, it is because it moved as the user clicks on it. Set this to `true` to avoid that extra click. *Only supported when using the DES Validation Framework.*

This solves the following problem:

When the user edits a control and immediately clicks on the button, the `onchange` event of the control fires, running validation. If validation removes error message and/or the `ValidationSummary`, the button may move. This happens before the button's `onclick` event, preventing that `onclick` event to run because the mouse button is no longer on the button.

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 frame 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 field 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**.

The Control To Extend can either be a specific data entry control, like a textbox, or a container of controls, like a Panel. When using a container, it captures the ENTER key for all controls it contains. Note that the container must generate an HTML tag that supports the client-side `onkeypress` event.

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 "".

Alert: This property requires a license that covers the Peter's Interactive Pages module.

- **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 the `NativeControlExtender`, 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**.

Alert: This property requires a license that covers the Peter's Interactive Pages module.

Hint Properties

See the “Interactive Hints” section of the **Interactive Pages User’s Guide** for details on Hints.

The Properties Editor shows these properties in the “Hint” category.

Alert: These properties requires a license that covers the Peter’s Interactive Pages module.

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

When blank, if the Control To Extend is using its **ToolTip** property, the **ToolTip** is used as the text of the hint unless you set the **HintManager.ToolTipAsHints** property to `False` or the **ToolTipUsesPopupViewName** property is used. In addition, the Control To Extend must be a data entry control. See the **HintManager.ToolTipAsHint** property in the **Interactive Pages User’s Guide** for details

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”.) 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 "".

- **HintHelpLookupID** (string) – Gets the value for **HintHelp** through the String Lookup System. (See “The String Lookup System”). 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 the global setting **DefaultSharedHintFormatterName**, which is set in the **Global Settings Editor**.

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.

ToolTip Properties

The Properties Editor shows these properties in the “Hint” category.

Alert: These properties require a license that covers the Peter’s Interactive Pages module.

- **ToolTip** (string) – Provides a tooltip for the Control to Extend. It can be used as a traditional tooltip or an Enhanced ToolTip by using **ToolTipUsesPopupViewName** and setting **HintManager.EnableToolTipsUsePopupViews** to **True**. (HintManager is accessed programmatically through **PeterBlum.DES.Globals.Page** and in the **PageManager** control.)

The Control to Extend may have its own **ToolTip** property. If it does, its value will be used when this is blank. If this is assigned it always overrides the **ToolTip** property on Control To Extend. In addition, you can use the String Lookup System to get the tooltip through **ToolTipLookupID**.

It defaults to "".

- **ToolTipLookupID** (string) – Gets the value for **ToolTip** through the String Lookup System. (See “The String Lookup System”.) The LookupID and its value should be defined within the String Group of **Hint**. 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.

Behavior Properties

The Properties Editor shows these properties in the “Behavior” category.

- **Enabled** (Boolean) – Determines if the control is used or not. When `true`, it is used. It defaults to `true`.
- **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”. It defaults to `false`.
- **ChangeMonitorEnables** (enum `PeterBlum.DES.ChangeMonitorEnablesSubmitControl`) – Used when the Control To Extend is a Button, LinkButton, or ImageButton. It determines if the button’s state is affected by the ChangeMonitor. When the ChangeMonitor, the button is disabled as the page is loaded. After the first edit, it becomes enabled.

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

- No - The button will not change its enable state.
- Yes - The button will change its enabled state.
- `CausesValidationIsTrue` - When the button's **CausesValidation** property is `true`, it will change its enabled state.
- `CausesValidationIsFalse` - When the button's **CausesValidation** property is `false`, it will change its enabled state.

It defaults to `ChangeMonitorEnablesSubmitControl.CausesValidationIsTrue`.

Alert: This property requires a license that covers the Peter’s Interactive Pages module.

- **ChangeMonitorGroups** (string) – When using the ChangeMonitor, the group name(s) defined here is marked changed when the Control To Extend is edited. See “ChangeMonitor” in the **Interactive Pages User’s Guide**.

If you have Validators evaluating the Control To Extend, those Validators already have a **Group** property which is used by the ChangeMonitor unless **ChangeMonitor.UseValidationGroups** is `false`. (**ChangeMonitor** is accessed programmatically through `PeterBlum.DES.Globals.Page` and in the **PageManager** control.)

Unless the validators do not specify the desired group, you can leave this blank.

The ChangeMonitor is enabled when **ChangeMonitor.Enabled** to `True` or the global setting **DefaultChangeMonitorEnabled** is `True` in the **Global Settings Editor**.

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 "".

Alert: This property requires a license that covers the Peter’s Interactive Pages module.

The LocalizableLabel Control

The `PeterBlum.DES.LocalizableLabel` control is a child of the `System.Web.UI.WebControls.Label` class. It extends the `Label` class with support for string lookup and localization. DES provides the “String Lookup System” where you can define the source of localized strings to be from `.resx` files, a database or another data source.

Use this control when building a web page that supports multiple languages. Often users develop separate pages, each with the text adjusted for the desired language. Other users prefer to have smart controls that change their own appearance.

`LocalizableLabel` can handle all text on the page. Yet, there is more to text localization. Here are several issues in making a page localizable.

- Often text on the page was typed directly into the HTML. This text must now be placed into ASP.NET controls, specifically the `LocalizableLabel`.
- Each language uses a different number of characters in the translation of your original text. Some will be much wider than the original. While HTML will widen the page or word-wrap the text as it grows, the layout of your page may no longer have a desirable appearance.

The DES’s String Lookup System can be made to use the data of various localization software products. You supply an event handler that handles a data lookup from the datasource of the localization software you are using. See “The String Lookup System”.

Click on any of these topics to jump to them:

- ◆ [Using the LocalizableLabel Control](#)
- ◆ [Adding the LocalizableLabel Control](#)
- ◆ [Properties of the LocalizableLabel Control](#)

Using the LocalizableLabel Control

The `System.Web.UI.WebControls.Label` is normally set up by defining text in the `Text` property or like this:

```
<asp:Label id=Label1 runat="server">Text goes here</asp:Label>
```

When you use the `LocalizableLabel`, the **Text** property remains and you use it for the default text. In addition, you assign a lookup ID to the **TextLookupID** property. The lookup ID identifies the string within your datasource of localized strings. See “The String Lookup System” for details.

Adding the LocalizableLabel Control

Visual Studio and VWD Design Mode Users

- Drag the LocalizableLabel control from the Toolbox onto your web form.
- Assign the **Text** and **TextLookupID** properties in the Properties Editor.

Text Entry Users

- In the <form> area, add the control itself:

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

- Add the default text between open and closing tags.

```
<des:LocalizableLabel id=LocalizableLabel1 runat="server">
  text here</des:LocalizableLabel>
```

- Add the **TextLookupID** property.

```
<des:LocalizableLabel id=LocalizableLabel1 runat="server"
  TextLookupID="lookupid" >
  text here</des:LocalizableLabel>
```

Programmatically Creating the Control

- Identify the control which you will add the LocalizableLabel control to its **Controls** collection. Like all ASP.NET controls, the LocalizableLabel 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 LocalizableLabel control class. The constructor takes no parameters.
- Assign the **ID** property.
- Add the LocalizableLabel control to the **Controls** collection.
- Assign the **Text** and **TextLookupID** properties.

In this example, the LocalizableLabel is created with an **ID** of "LocalizableLabel1". It is added to Placeholder1.

[C#]

```
PeterBlum.DES.LocalizableLabel vLabel = new PeterBlum.DES.LocalizableLabel();
vLabel.ID = "LocalizableLabel1";
vLabel.Text = "value";
vLabel.TextLookupID = "lookupID";
Placeholder1.Controls.Add(vLabel);
```

[VB]

```
Dim vLabel As PeterBlum.DES.LocalizableLabel = _
  New PeterBlum.DES.LocalizableLabel()
vLabel.ID = "LocalizableLabel1"
vLabel.Text = "value"
vLabel.TextLookupID = "lookupID"
Placeholder1.Controls.Add(vLabel)
```

Properties of the LocalizableLabel Control

The LocalizableLabel control is subclassed from `System.Web.UI.WebControls.Label`. It inherits the properties, methods and events of `Label`. See `System.Web.UI.WebControls.Label` for details.

- **Text** (string) – Inherited from `System.Web.UI.WebControls.Label`, it supplies the default text that is used when the **TextLookupID** does not find a match in the datasource. If you can be assured of a match, you can leave this blank. It supports HTML tags.

Reminder: If your page should follow the XHTML standard, make sure your tags conform to XHTML.

- **TextLookupID** (string) – Gets the value for **Text** through The String Lookup System. The LookupID and its value should be defined within the String Group of Labels. If no match is found OR this is blank, **Text** 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 "".

- **AssociatedControlID** (string) – An ID to another control associated with this label.

Use the **AssociatedControlID** property to associate a Label control with another server control on a Web form. When a Label control is associated with another server control, its attributes can be used to extend the functionality of the associated control. You can use the Label control as a caption for another control, or you can set the tab index or hot key for an associated control.

When the **AssociatedControlID** property is set, the Label control renders as an HTML label element, with the for attribute set to the ID property of the associated control. You can set other attributes of the label element using the Label properties. For example, you can use the Text and AccessKey properties to provide the caption and hot key for an associated control.

Credit: Some of this text is adapted from Microsoft's Help text on this property.

Use it when the Associated is in the same or any parent naming container. If assigned to an unknown controlID or one in an incorrect naming container, an exception will be thrown at runtime.

It defaults to "".

- **AssociatedControl** (Control) – A reference to a Associated control. It is an alternative to **AssociatedControlID** that allows the control to be anywhere on the page instead of the same naming container as the Label.

Global Settings Editor and custom.DES.config File

DES lets you set a number of site-wide defaults using the **Global Settings Editor** application. See “Using the Global Settings Editor”. These settings are stored in the **custom.des.config** file that every web application must have. By default, this file is located in the **[web application]DES** folder. A companion file, **des.config**, contains the factory global settings and should not be edited.

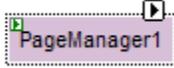
When the web application first requests global data from the **custom.des.config** file, it loads the data into the global objects **PeterBlum.DES.Globals** and **PeterBlum.DES.StringLookup**. Each value is assigned to a static/shared property, usually with an identical name to the property shown in the **Global Settings Editor**. You can programmatically assign these defaults within your **Global.asax** file. See “Programmatically Assigning Globals”.

Click on any of these topics to jump to them:

- ◆ Using the Global Settings Editor
 - Debugging the Global Settings Editor Properties
- ◆ Programmatically Assigning Globals
 - Adding Globals After Config Files Load
- ◆ DES.config and Custom.DES.config File

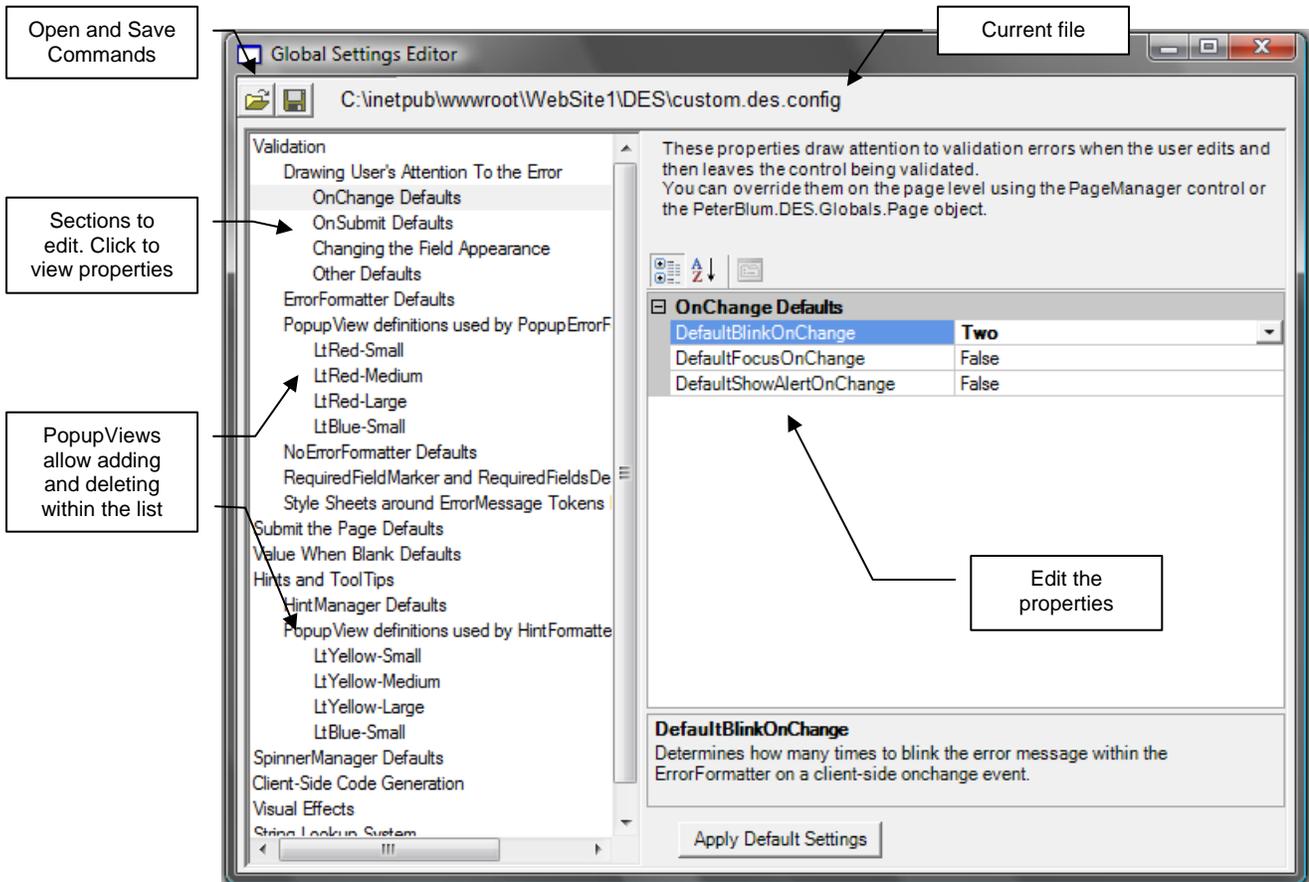
Using the Global Settings Editor

There are several ways to start the **Global Settings Editor**:

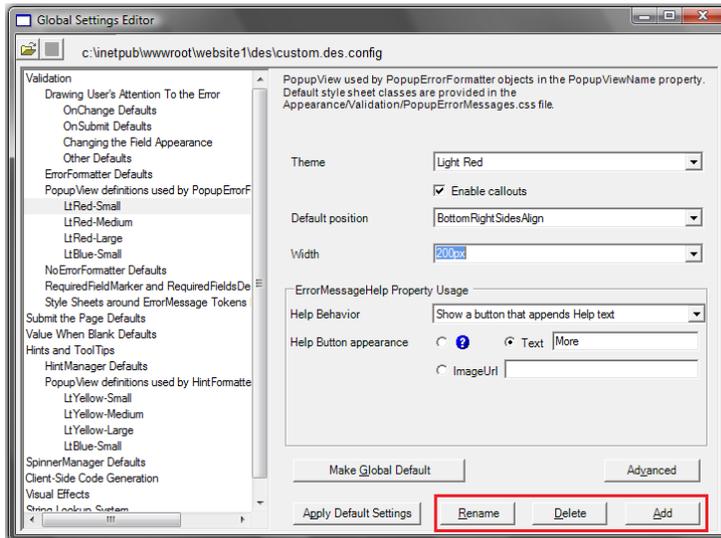
- From the SmartTag of the PageManager control in design mode. 
- From the Start menu, select Peter’s Data Entry Suite. Then select **Global Settings Editor**.
- From the [DES installation folder] folder.
- If you are using Visual Studio, you can add it to your **Tools** menu using the **External Tools** command. Locate the program in the **[DES installation folder]** folder.

Once launched, you select the **custom.des.config** file. Then click on the topics in the left view to reveal their properties. When finished, click the Save button in the toolbar.

Note: If you edit the custom.DES.config file of a web application that has been started, the changes will have no effect. You must restart your web application.



While most sections display a Property Editor, the PopupView definitions used by PopupErrorFormatters and HintFormatters work differently. These features allow you to add PopupView definitions. Each of these has a unique name that you use in the PopupViewName properties on PopupErrorFormatters and HintFormatters.



- To add, click on the PopupView definitions heading and click the **Add** button.
- To delete, click on the PopupView name and click the **Delete** button.
- To rename, click on the PopupView name and click the **Rename** button.
- The PopupView editor does not show every available property. Click the **Advanced** button to see all properties.

Debugging the Global Settings Editor Properties

If you are uncertain that the values set in the **Global Settings Editor** are loaded correctly, set up a special querystring parameter, as described in the “Exploring The Current Settings”. Then point your browser to a page in your web application with the addition of your querystring parameter and the command “globals”. For example, if your querystring parameter is “desdebug”:

```
http://www.myapp.com?desdebug=globals
```

It will output a page listing all properties on the **PeterBlum.DES.Globals** property, which includes those from the **custom.des.config** and **web.config** files.

Alternatively, use the `PeterBlum.DES.Globals.DescribeProperties()` method. It returns a string that you can assign to a `Label` or `LiteralControl` control to show on the page or you can set up `<@ Page Trace="true" >` and output them using `Page.Trace.Write()`.

```
PeterBlum.DES.Globals.DescribeProperties(showHTML)
```

showHTML

Pass `true` to format the text in HTML format (as a table) and `false` to format it as a carriage return delimited set of lines useful to output to a file or other system that cannot use HTML.

Call it within the `Page_Load()` method, like this.

```
DebugLabel1.Text = PeterBlum.DES.Globals.DescribeProperties(true)
Page.Trace.Write(PeterBlum.DES.Globals.DescribeProperties(false))
```

Programmatically Assigning Globals

If you want to apply values programmatically, do it in the **Global.asax** file using the `Application_Start()` method. Just know the name of the property. Then assign it to either **PeterBlum.DES.Globals** or **PeterBlum.DES.StringLookup**, as appropriate.

Note: The first time you access a property on `PeterBlum.DES.Globals`, it will load the values from `custom.DES.config`.

Here is an example where the default for **PeterBlum.DES.Globals.Page.ShowAlertOnChange** is set in the `Application_Start()` method of **Global.asax**. The global property name is **DefaultShowAlertOnChange**.

[C#]

```
protected void Application_Start(Object sender, EventArgs e)
{
    PeterBlum.DES.Globals.DefaultShowAlertOnChange = true;
}
```

[VB]

```
Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
    PeterBlum.DES.Globals.DefaultShowAlertOnChange = True
End Sub
```

Adding Globals After Config Files Load

While assigning property settings to the `PeterBlum.DES.Globals` class can be done within `Application_Start()`, there are several collection-type properties within the **des.config** file for which you may want to add items. This should happen after the config files have been loaded because each time they load, they clear the collection-type properties.

DES defines the `PeterBlum.DES.ConfigFile` class to manage the settings loaded from the various config files.

These properties include:

Property name in <code>PeterBlum.DES.ConfigFile</code>	Collection of (class)	Defined in this section of <code>custom.des.config</code>
DataTypes	Subclasses of <code>DESTypeConverters</code>	<DataTypes>
ErrorFormatters	Subclasses of <code>BaseErrorFormatter</code>	<ErrorFormatters>
Conditions	Subclasses of <code>BaseCondition</code>	<Conditions>
ErrorMessagePopupViews	<code>ErrorMessagePopupView</code>	<ErrorMessagePopupViews>
HintPopupViews	<code>HintPopupView</code>	<HintPopupViews>
CreditCards	<code>CreditCardDescription</code>	<CreditCards>
ThirdPartyControls	<code>ThirdPartyControlDef</code>	<ThirdPartyControls> See Using Third Party Controls.pdf
RequiredSelectionControls	<code>RequiredSelectionControl</code>	<RequiredSelectionControls>

Use the **PeterBlum.DES.ConfigFile.ConfigFilesLoaded** event to invoke your own method that can add to any of these lists. It is passed the `PeterBlum.DES.ConfigFile` object. Here is the delegate definition:

[C#]

```
delegate void ConfigFilesLoadedEventHandler(PeterBlum.DES.ConfigFile pConfigFile);
```

[VB]

```
Delegate Sub ConfigFilesLoadedEventHandler( _
    ByVal pConfigFile As PeterBlum.DES.ConfigFile)
```

This is how you attach to the **ConfigFilesLoaded** event:

[C#]

```
protected void Application_Start(Object sender, EventArgs e)
{
    PeterBlum.DES.ConfigFile.ConfigFilesLoaded +=
        new PeterBlum.DES.ConfigFilesLoadedEventHandler(MyConfigFilesLoaded);
}

protected void MyConfigFilesLoaded(PeterBlum.DES.ConfigFile pConfigFile)
{
    // Modify the collection-type properties in pConfigFile here
    // For most, create an instance of the object
    // then pass it to the Add() method on the property.
    // For ThirdPartyControls, see the Using Third Party Controls.pdf
    // for available methods on the pConfigFile.ThirdPartyControls method.
}
```

[VB]

```
Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
    AddHandler PeterBlum.DES.ConfigFile.ConfigFilesLoaded _
        AddressOf MyConfigFilesLoaded
End Sub

Protected Sub MyConfigFilesLoaded(ByVal pConfigFile As PeterBlum.DES.ConfigFile)
    ' Modify the collection-type properties in pConfigFile here
    ' For most, create an instance of the object
    ' then pass it to the Add() method on the property.
    ' For ThirdPartyControls, see the Using Third Party Controls.pdf
    ' for available methods on the pConfigFile.ThirdPartyControls method.
End Sub
```

DES.config and Custom.DES.config File

DES provides two XML configuration files that allow you to customize the design mode and runtime default values. Both of these files are installed into the [web application root]\DES\ folder.

- **des.config** – Settings defined by PeterBlum.com. You do not edit this file. It is replaced on each new release of DES.
- **custom.des.config** – Settings that you edit, usually with the **Global Settings Editor**.

By having two files describing the configuration, any updates to **des.config** will not require that you replace or merge your own edits. You simply get a new **des.config** file and leave **custom.des.config** untouched.

These configuration files contain these sections. Here is how to edit them:

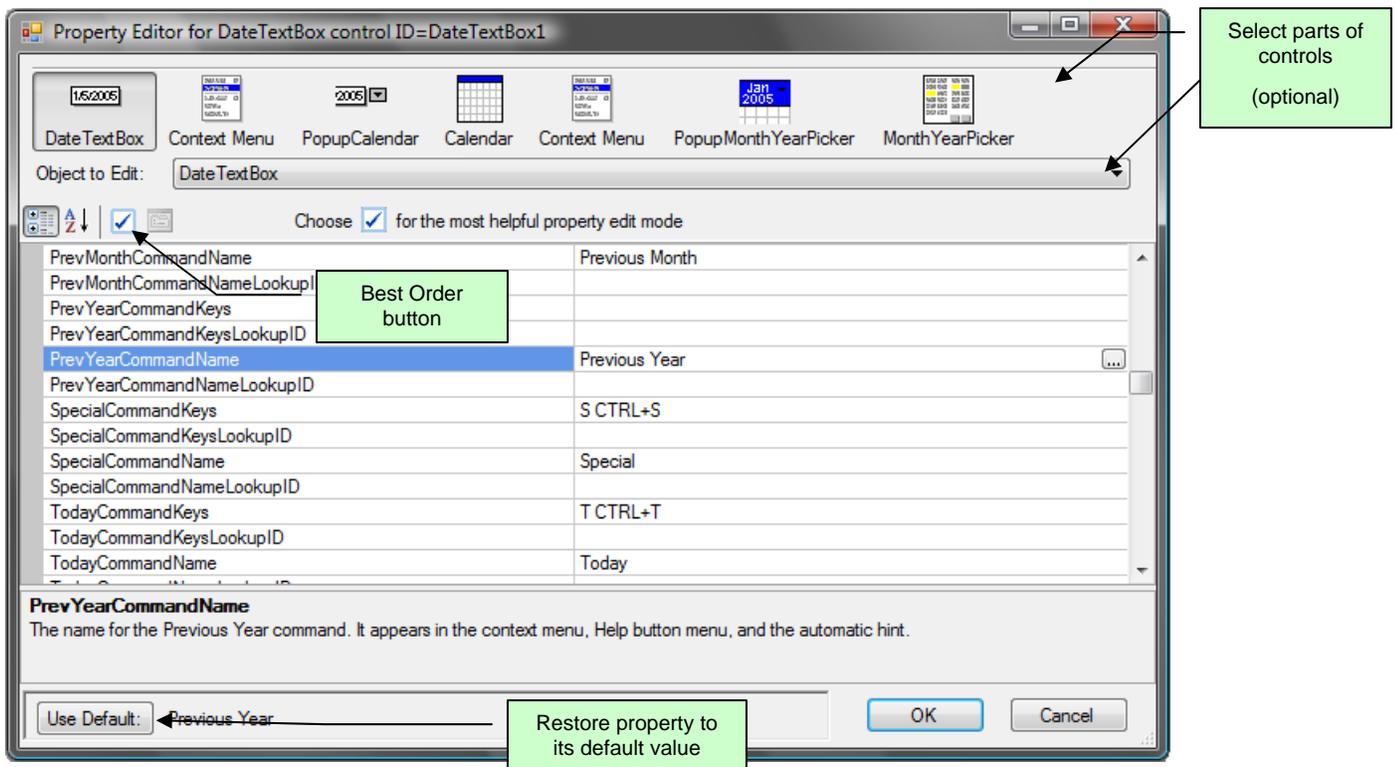
- Global Properties (\Configuration\GlobalProperties) – These values are edited by the **Global Settings Editor**. They are assigned to static/shared properties on **PeterBlum.DES.Globals**.
- String Lookup (\Configuration\StringLookup) - These values are edited by the **Global Settings Editor**. They are assigned to static/shared properties on **PeterBlum.DES.StringLookup**.
- Regular Expression Patterns (\Configuration\RegExPatterns) – A list of the Regular expressions found in the RegExValidator's editor for its **Expressions** property. Each expression has a unique name.
- DataTypes (\Configuration\DataTypes) – A list of data types (PeterBlum.DES.DESTypeConverter subclasses) that appear in the editor for the **Data Type** property of various Validator controls. Each DESTypeConverter has a unique name that appears in the editor.
- Error Formatters (\Configuration\ErrorFormatters) – A list of Error Formatters (PeterBlum.DES.BaseErrorFormatter subclasses) that appear in the editor for the **ErrorFormatter** property of Validator controls. Each Error Formatter has a unique name that appears in the editor.
- Conditions (\Configuration\Conditions) – A list of Conditions (PeterBlum.DES.BaseCondition subclasses) that appear in the editor for the **Enabler** property of Validator controls and within the **Conditions** property of the MultiConditionValidator and CountTrueConditionValidator controls. Each Condition has a unique name that appears in the editor.
- ErrorMessagePopupViews (\Configuration\ErrorMessagePopupViews) – A list of PopupView definitions used by the PopupErrorFormatters. These values are edited by the **Global Settings Editor**.
- HintPopupViews (\Configuration\HintPopupViews) – A list of PopupView definitions used by the HintFormatters. These values are edited by the **Global Settings Editor**.
- CreditCards (\Configuration\CreditCards) – A list of credit card patterns and rules for use with the CreditCardNumberValidator and CreditCardNumberCondition.
- Third Party Controls (\Configuration\ThirdPartyControls) - Any third party controls that act like Microsoft's TextBox, RadioButton, CheckBox, ListBox or DropDownList but are not subclassed from those controls can still be used in DES by defining them here. Once defined, DES will list them in the ControlIDToEvaluate (and similar properties) in the Properties Editor and various conditions that support the Microsoft controls will support the third party control. See "Supporting Third Party Data Entry Controls" in the **Installation Guide**.
- Get Child Methods (\Configuration\GetChildMethods) – Used when you develop custom controls that have multiple controls representing one value, such as a list of radio buttons. This defines the client-side JavaScript function that will gather data from those controls.

Expanded Properties Editor

In design mode, the Properties Editor is used to change property values. Because many DES controls are really several controls combined together, an **Expanded Properties Editor** is now available. It offers these enhancements over the Visual Studio Properties Editor:

- Quickly jump between the properties for each part of the selected control. The picture below shows DateTextBox with the toolbar along the top showing the parts. *Note: The toolbar and Object To Edit elements are not shown on many DES controls.*
- The normal Properties Editor lists properties sorted by their names and category names. This ordering is not effective because it does not group related properties together and in an order that makes sense. The Expanded Properties Editor provides an ordering that is better for setting up the control by clicking the (Best Order) button.
- There are numerous properties. How do you know which ones are important or not often used? When you use the (Best Order) button, it can show properties by their importance by using the buttons **Recommended Properties** and **Properties Assistant**.
- You can quickly return a single property to its default value by selecting it and clicking the **Use Default** button.

You can select it from the context/task menu on the control, from the Properties Pages icon at the top of Visual Studio's Properties Editor, and from a link at the bottom of Visual Studio's Properties Editor.



Using the Expanded Properties Editor

The toolbar and dropdownlist at the top let you switch to the various controls that are part of this control. Shown above is the DateTextBox. If you notice two Context Menus, it's because both the DateTextBox and its Calendar provide Context Menus.

The most powerful feature of this window is the "Best Order" button immediately above the property names. Click it to have the properties and categories listed in the order recommended by PeterBlum.com for setting up the control.

Once the Best Order button is selected, the toolbar looks like this:



All properties have been assigned one of these states: Required, Recommended, Sometimes used, or Rarely used. Click the **Recommended Properties** button to show only the Required and Recommended properties. This is a very good way to set up a newly added control. If you want to customize which states are shown, use the **Properties Assistant** button.

Finally, along the bottom are the **Use Default** button and the default value shown to its right. This is a quick way to restore the individual property to its default.

Using These Controls with AJAX

AJAX is a technology that allows modern browsers to communicate with your server side code without a postback. It typically is used to update values through DHTML – the object oriented framework of the browser. The page developer may replace the value of a single attribute on an HTML tag, such as the text shown on a textbox or replace a block the page's HTML.

AJAX is packaged within many third party products, including Microsoft ASP.NET AJAX, Telerik RadControls (“RadAjax”), and MagicAjax. Each of these products has the capability to replace a block of the page's HTML with any web controls you desire.

When those controls are within **Peter's Data Entry Suite**, you must take additional actions using the `PeterBlum.DES.AJAXManager` class. What follows is a brief overview of these actions. They are detailed in each section associated with AJAX framework you are using.

ALERT: *You are virtually guaranteed **JavaScript errors** and **lack of functionality** if you do not implement these directions correctly.*

Click on any of these topics to jump to them:

- ◆ [Using Microsoft ASP.NET AJAX](#)
- ◆ [Using Telerik RadAjax](#)
- ◆ [Using other AJAX-enabled Telerik controls](#)
- ◆ [Using Infragistics AJAX-enabled Controls](#)
- ◆ [Using ComponentArt CallBack](#)
- ◆ [Using MagicAjax](#)
- ◆ [Using other AJAX Products](#)
- ◆ [AJAXManager Properties](#)
- ◆ [Other AJAXManager Methods](#)
- ◆ [Analyzing the InAJAXUpdate Properties on DES Controls](#)

ALERT: *Not every third party AJAX product will work with Peter's Data Entry Suite, even after following the directions here. Some may require an entirely different approach. For example, Anthem requires that you subclass each control you intend to update the HTML. PeterBlum.com will not be developing custom code for these products.*

Using Microsoft ASP.NET AJAX

<http://ajax.asp.net>

Microsoft ASP.NET AJAX framework supplies the UpdatePanel control to replace HTML. You need to apply these steps if any DES control is inside an UpdatePanel or refers to a control inside the UpdatePanel.

Examples follow these steps.

ALERT: *You are virtually guaranteed JavaScript errors and lack of functionality if you do not implement these steps correctly.*

1. Make sure the ScriptManager control is on the page.
2. Set **ScriptManager.EnablePartialRendering** to `true` either in the ASP.NET declaration or in the Page's PreInit event.
3. Tell DES that you are using Microsoft AJAX with its controls on this page.

When using the PageManager Control:

Set the **AJAXFramework** property to `MicrosoftAJAX`. Leave the **AJAXControlID** property unassigned.

```
<des:PageManager runat="server" AJAXFramework="MicrosoftAJAX" />
```

Programmatically:

Call the `UsingMicrosoftAJAX()` method in `Page_Load()`. The method is shown here:

```
PeterBlum.DES.AJAXManager.UsingMicrosoftAJAX()
```

ALERT: *For any code that uses the `PeterBlum.DES.AJAXManager`, it must run every time `Page_Load()` is called. Do not nest it inside of an IF statement like `If IsPostBack = False Then`.*

4. Identify which DES controls are participating in the AJAX update. They need to transfer data to the client-side during each callback. Aside from those inside of an UpdatePanel, include those that point to controls inside of UpdatePanels. Examples: Validators pointing to data entry controls, Enabler conditions, FieldStateControllers, CalculationControllers, and NativeControlExtenders.

The challenge is that if you include controls that are not part of the callback, the callback will have poorer performance because there is more data transferred and processed. To get the best performance, more work is involved in setup.

If you don't include controls that are part of the callback, after the callback those controls either will not operate correctly or will generate javascript errors. So some care is needed.

There are three approaches:

- **Include all controls in the callback**
 - Easy setup, no operating problems after callback.
 - Poorest performance where data transfer is at the maximum. Performs very well when most or all of the DES controls are involved in the callback.
- **Good Performance** – *This is the default setup.*
 - Easy but manual assignment of the **InAJAXUpdate** property is required for controls outside of UpdatePanels that connect with controls involved in AJAX.
 - Data transfer is minimized although the server does more work.
 - Incorrect setup results in controls malfunctioning.
- **Best Performance**
 - Must identify every control involved in the callback
 - Data transfer is minimized and the server does less work.
 - Incorrect setup results in controls malfunctioning.

They are discussed below.

Include All Controls In the Callback

Choose this approach:

- when most or all DES controls are involved in the Callback
- when you have encountered malfunctions after a callback to see if they go away. (You will then turn it off and set `InAJAXUpdate = true` on the controls with the problems.)

When using the PageManager control:

Set the **AllInAJAXUpdate** property to `true`.

```
<des:PageManager runat="server"
    AJAXFramework="MicrosoftAJAX" AllInAJAXUpdate="True" />
```

Programmatically:

Set the **PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate** property to `true` in `Page_Load()`. This will tell all DES controls to transmit their data during each AJAX callback.

```
PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate = true
```

See “Example 1 – Using the AllInAJAXUpdate property”.

Good Performance

This is the default setup. Performance is determined by how much data is transmitted during the callback and how much work the server has to do to create that data. The best performance is to set the **InAJAXUpdate** property to `true` on every control involved in the callback, as described below in “Best Performance”, below. That requires some work on your part.

This option provides the best performance for data transmitted at the expense of the server performance. The server sets **InAJAXUpdate** after searching for the UpdatePanel as a container control. That searching is where the server does more work.

Using this method:

- By default, this method is active. If you have previously set **AllInAJAXUpdate** to `true` (see above) or **SmartSetInAJAXUpdate** to `false`, switch their value. (Those properties are on the PageManager control or the **PeterBlum.DES.Globals.Page.AJAXManager** object.)
- For any DES controls outside of UpdatePanels that point to controls inside of the UpdatePanels, you must set their **InAJAXUpdate** property to `true`, as shown under “Optimized performance”, below. Examples: Validators pointing to data entry controls, Enabler conditions, FieldStateControllers, CalculationControllers, and NativeControlExtenders.
- Feel free to set **InAJAXUpdate** to `true` on any DES controls that are part of the callback to avoid having the CPU search for the UpdatePanel. For any you don’t set, DES will do the detection of the UpdatePanel.

BEST PERFORMANCE DESCRIBED ON THE NEXT PAGE

Best Performance

By setting **InAJAXUpdate** to `true` on exactly those DES controls involved in the callback, transmission time and CPU time are reduced.

Using this method:

- Set to `false` the two properties that make setup easier, but less optimal: **AllInAJAXUpdate** and **SmartSetInAJAXUpdate**. **AllInAJAXUpdate** defaults to `false`, so you probably won't need to do anything to it.

When using the *PageManager* control:

Set the **SmartSetInAJAXUpdate** property to `false`.

```
<des:PageManager runat="server" AJAXFramework="MicrosoftAJAX"
  SmartSetInAJAXUpdate="False" />
```

Programmatically:

Set the **PeterBlum.DES.AJAXManager.Current.SmartSetInAJAXUpdate** property to `false` in `Page_Load()`. This will tell all DES controls to transmit their data during each AJAX callback.

```
PeterBlum.DES.AJAXManager.Current.SmartSetInAJAXUpdate = false
```

- Every DES control has the **InAJAXUpdate** property. Set it to `true` on those that are in the `UpdatePanel`, or are connected to controls inside the `UpdatePanel`. For example:

```
<des:RequiredTextValidator InAJAXUpdate="true" [other properties] />
<des:TextBox InAJAXUpdate="true" [other properties] />
<des:FieldStateController InAJAXUpdate="true" [other properties] />
<des:Button InAJAXUpdate="true" [other properties] />
<des:CalculationController InAJAXUpdate="true" [other properties] />
```

- To analyze how the **InAJAXUpdate** property on DES's controls have been set up, see "Analyzing the InAJAXUpdate Properties on DES Controls".

See "Example 2 – Identify individual DES Controls for Best Performance".

Visual Studio and Visual Web Developer design mode suggestions:

- Select all DES controls that are in the AJAX update. The Properties Editor will display all properties they have in common, including **InAJAXUpdate**. Set it to `true` to update all at once.
- ASP.NET 2 only.* Each DES control has a SmartTag with the setting **Involved in an AJAX Update**. Mark its checkbox.

To let the computer do some of the work, use the

`PeterBlum.DES.AJAXManager.SetChildrenInAJAXUpdate()` method to update a container control, like a panel, so that all of its child DES controls have their **InAJAXUpdate** property set to `true`. If any DES control points to another control outside of the container, its **InAJAXUpdate** property will also be set to `true`. However, if there is a DES control outside the container that is not pointed to by a control inside the container, it will still need you to set **InAJAXUpdate**. See "PeterBlum.DES.AJAXManager.SetChildrenInAJAXUpdate".

CONTINUE TO THE NEXT PAGE FOR STEPS 5 AND 6

5. If a type of DES control or feature is not initially added to page but will later be added during a callback, it needs to be identified when the page is initially requested (**Page.IsPostBack** = false). Examples include a DataGrid or GridView that switches to edit mode with textboxes and validators; the MultiView; and the Wizard which often does not have data entry controls on the first step.

When using the PageManager control:

Using the chart on the next page, set the desired property to true on the **PageManager.PreLoadForAJAX** property. For example:

```
<des:PageManager runat="server" PreLoadForAJAX-Validators="True"
    PreLoadForAJAX-TextBoxes="True" />
```

Programmatically:

Call `PeterBlum.DES.AJAXManager.PreregisterForAJAX()`.

You pass the enumerated type `PeterBlum.DES.AJAXManager.FeatureList`, whose values are listed in the chart below. For example:

```
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.Validators)
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.TextBoxes)
```

Note: It is safe to call PreregisterForAJAX() even when the controls are shown when the page is first generated. All it does is guarantee scripts are added to the page. If you call it when the controls will never be shown, it adds the overhead of loading script files and setting up small global objects.

The `PreregisterForAJAX()` method also accepts an array of `FeatureList` values like this:

[C#]

```
using PeterBlum.DES;
...
AJAXManager.FeatureList[] vFeatures =
    {AJAXManager.FeatureList.Validators, AJAXManager.FeatureList.TextBoxes};
AJAXManager.PreregisterForAJAX(vFeatures);
```

[VB]

```
Imports PeterBlum.DES
...
Dim vFeatures() As AJAXManager.FeatureList = New AJAXManager.FeatureList() _
    {AJAXManager.FeatureList.Validators, AJAXManager.FeatureList.TextBoxes}
AJAXManager.PreregisterForAJAX(vFeatures)
```

See “Example 3 – Controls added during Callback”.

CHART OF FEATURES IS ON THE NEXT PAGE

AJAXManager.FeatureList values and PageManager.PreLoadForAJAX properties	
Validators	Use when no validator is on the initial page but will be added during a callback.
ValidationSummary	Use when no ValidationSummary is on the initial page but will be added during a callback.
CombinedErrorMessages	Use when a CombinedErrorMessages control is added on a callback.
TextBoxes	Use when no DES TextBox is on the initial page but will be added during a callback. This includes TextBox, FilteredTextBox, IntegerTextBox, DecimalTextBox, PercentTextBox, and CurrencyTextBox. Also consider Hints, AutoPostBack.
MultiSegmentDataEntry	Use when no MultiSegmentDataEntry control is on the initial page but will be added during a callback. Also consider Hints.
Hints (Also when using ToolTips)	Use when no hints are on the initial page but will be used during a callback. This also applies when using the hint properties on DES's textboxes and MultiSegmentDataEntry controls. By default, ToolTips are converted to hints. When using ToolTips, you need to preload the Hints feature or to turn off the conversion of ToolTips to hints with the ToolTipsAsHints property on the PageManager control or PeterBlum.DES.Globals.Page . See "Properties on the PeterBlum.DES.Globals.Page.HintManager Property" topic in the Interactive Pages Guide .
DateTextBoxes *	Use when no DateTextBox, AnniversaryTextBox, or MonthYearTextBox is on the initial page but will be added during the callback. Also consider Hints, AutoPostBack.
TimeTextBoxes *	Use when no TimeOfDayTextBox or DurationTextBox is on the initial page but will be added during the callback. Also consider Hints, AutoPostBack.
Calendar *	Use when no Calendar is on the initial page but will be added during the callback.
MultiSelectionCalendar *	Use when no MultiSelectionCalendar is on the initial page but will be added during the callback.
MonthYearPicker *	Use when no MonthYearPicker or PopupMonthYearPicker is on the initial page but will be added during the callback.
TimePicker *	Use when no TimePicker or PopupTimePicker is on the initial page but will be added during the callback.
SpecialDates	Use when no SpecialDates control is on the initial page but will be added during the callback.
ContextMenu	Use when no ContextMenu control is on the initial page but will be added during the callback.
FieldStateControllers	Use when no type of FieldStateController is on the initial page but will be added during a callback. This includes FieldStateController, MultiFieldStateController, FSCOnCommand, and MultiFSCOnCommand.
CalculationControllers	Use when no CalculationController is on the initial page but will be added during a callback.
TextCounter	Use when no TextCounter is on the initial page but will be added during the callback.
AutoPostBack	Use when you validate on AutoPostBack.
ChangeMonitor	Use when the ChangeMonitor is added to the page after postback.
SubmitControls	Use when no Button, LinkButton, or ImageButton is on the initial page but will be added during the callback.
DisableOnSubmit	Use when no Button uses the DisableOnSubmit feature on the initial page but will be used during a callback.
Popups	Use when you create a control using the Popup code.

* The next step applies in these cases.

6. For any of the following items from the above chart: `DateTextBoxes`, `TimeTextBoxes`, `Calendar`, `MultiSelectionCalendar`, `MonthYearPicker`, and `TimePicker`, their popup features must be preloaded during the initial page load, even though the control itself is not shown until the AJAX update.

Follow these instructions for *each type of control* that was identified:

- Create a new instance of the control to the page. Put it just after the `<form>` tag in the web form (aspx) file or at the beginning of the `UserControl`. *It should **not** be inside the `UpdatePanel`.* This will avoid having it enclosed in another web control that may have its **Visible** property set to `false`.

```
<form id="form1" runat="server" >
  <des:DateTextBox id="DummyDTB" runat="server" />
  <asp:othercontrols here />
```

Make sure this control is always added to the page, even if you are not sure if same type of control will actually be added during a callback. **WARNING:** *It must be added to the Page's Control tree before any of the same type controls that you plan to show in the AJAX panel because its `PreRender` method must run first. Otherwise, you will get JavaScript errors after the callback.*

- Set up the popup elements on this control the way you want to see them when the control is shown in the browser. These popups will be written out when the page is first generated.

```
<des:DateTextBox id="DummyDTB" runat="server" >
  <PopupCalendar>
    <Calendar propertyname="value" propertyname2="value" />
  </PopupCalendar>
</des:DateTextBox>
```

- Call the new control's `PreLoadForAJAX()` method in `Page_Load()`.

```
DummyDTB.PreLoadForAJAX()
```

The call to `PreLoadForAJAX()` prevents this control from appearing on the page, but DES will use it to preload the HTML and scripts of its popups.

See "Example 4 – Control with Popups first created during Callback".

Example 1 – Using the AllInAJAXUpdate property

Suppose you have an UpdatePanel with a DES TextBox, an associated RequiredTextValidator, and a DES Button. You choose the **AllInAJAXUpdate** property because these are the only DES controls on the page.

```
<asp:ScriptManager id="ScriptManager1" runat="server"
    EnablePartialRendering="True">
</asp:ScriptManager>
<asp:UpdatePanel id="UpdatePanel1" runat="server">
    <ContentTemplate>

        <des:TextBox id="TextBox1" runat="server" />&nbsp;
        <des:RequiredTextValidator id="RTV1" runat="server"
            ControlIDToEvaluate="TextBox1" ErrorMessage="Required" /><br />
        <des:Button id="Submit" runat="server" Text="Submit" />

    </ContentTemplate>
</asp:UpdatePanel>
```

Using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
    AJAXFramework="MicrosoftAJAX" AllInAJAXUpdate="True" >
</des:PageManager>
```

Programmatically:

This code goes in Page_Load():

[C#]

```
PeterBlum.DES.AJAXManager.UsingMicrosoftAJAX();
PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate = true;
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingMicrosoftAJAX()
PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate = True
```

Example 2 – Identify individual DES Controls for Best Performance

Suppose you have an UpdatePanel with a DES TextBox, an associated RequiredTextValidator, and a DES Button. You want to use the **InAJAXUpdate** property on each DES control in the UpdatePanel because there are other DES controls on the page.

*Note: To analyze how the **InAJAXUpdate** property on DES's controls have been set up, see "Analyzing the InAJAXUpdate Properties on DES Controls".*

```
<asp:ScriptManager id="ScriptManager1" runat="server"
    EnablePartialRendering="True">
</asp:ScriptManager>
<asp:UpdatePanel id="UpdatePanel1" runat="server">
    <ContentTemplate>
        <des:TextBox id="TextBox1" runat="server" InAJAXUpdate="true" />&nbsp;
        <des:RequiredTextValidator id="RTV1" runat="server" InAJAXUpdate="true"
            ControlIDToEvaluate="TextBox1" ErrorMessage="Required" /><br />
        <des:Button id="Submit" runat="server" Text="Submit" InAJAXUpdate="true" />
    </ContentTemplate>
</asp:UpdatePanel>
```

Using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
    AJAXFramework="MicrosoftAJAX" >
</des:PageManager>
```

Programmatically:

This code goes in Page_Load ():

[C#]

```
PeterBlum.DES.AJAXManager.UsingMicrosoftAJAX();
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingMicrosoftAJAX()
```

Example 3 – Controls added during Callback

Suppose you have an UpdatePanel with a DES TextBox, an associated RequiredTextValidator, and a DES Button. The TextBox and Validator controls are inside of a Panel which is **Visible=false** until the DES Button sets it to **Visible=true** during a callback.

```
<asp:ScriptManager id="ScriptManager1" runat="server"
    EnablePartialRendering="True">
</asp:ScriptManager>
<asp:UpdatePanel id="UpdatePanel1" runat="server">
    <ContentTemplate>

        <asp:Panel id="Panel1" runat="server" Visible="false">
            <des:TextBox id="TextBox1" runat="server" InAJAXUpdate="true" />&nbsp;
            <des:RequiredTextValidator id="RTV1" runat="server" InAJAXUpdate="true"
                ControlIDToEvaluate="TextBox1" ErrorMessage="Required" /><br />
        </asp:Panel>
        <des:Button id="Show" runat="server" Text="Submit" InAJAXUpdate="true"
            Click="Show_ButtonClick" />

    </ContentTemplate>
</asp:UpdatePanel>
```

Using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
    AJAXFramework="MicrosoftAJAX"
    PreLoadForAJAX-Validators="True" PreLoadForAJAX-TextBoxes="True">
</des:PageManager>
```

Programmatically:

This code goes in Page_Load():

[C#]

```
PeterBlum.DES.AJAXManager.UsingMicrosoftAJAX();
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.Validators);
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.TextBoxes);
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingMicrosoftAJAX()
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _
    PeterBlum.DES.AJAXManager.FeatureList.Validators)
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _
    PeterBlum.DES.AJAXManager.FeatureList.TextBoxes)
```

This code goes in Show_ButtonClick():

[C#]

```
Panel1.Visible = true;
```

[VB]

```
Panel1.Visible = True
```

Example 4 – Control with Popups first created during Callback

Suppose you have an UpdatePanel with GridView that shows a DateTextBox in edit mode. Until the GridView is in edit mode, the <EditItemTemplate> is not created, requiring a dummy DateTextBox to preload the popups. The dummy DateTextBox's popups must have the same properties as the actual DateTextBox. In this case, **CssClass** is set to a custom style sheet class. *Reminder: The Dummy control must not be inside the UpdatePanel. Try to get it just below the <form> tag.*

```
<des:DateTextBox id="DummyDTB" runat="server" >
  <PopupCalendar>
    <Calendar CssClass="MyCalendarClass" />
  </PopupCalendar>
</des:DateTextBox>

<asp:ScriptManager id="ScriptManager1" runat="server"
  EnablePartialRendering="True">
</asp:ScriptManager>
<asp:UpdatePanel id="UpdatePanel1" runat="server">
  <ContentTemplate>

    <asp:GridView id="GridView1" runat="server" OnRowCreated="GridView1_RowCreated" >
    <Columns>
      <des:CommandField ShowEditButton="True" />
      <asp:TemplateField>
        <ItemTemplate>...</ItemTemplate>
        <EditItemTemplate>

          <des:DateTextBox id="DateTextBox1" runat="server" InAJAXUpdate="true" >
            <PopupCalendar>
              <Calendar CssClass="MyCalendarClass" />
            </PopupCalendar>
          </des:DateTextBox>

        </EditItemTemplate>
      </asp:TemplateField>
    </Columns>
  </asp:GridView>

  </ContentTemplate>
</asp:UpdatePanel>
```

Using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
  AJAXFramework="MicrosoftAJAX"
  PreLoadForAJAX-DateTextBoxes="True" >
</des:PageManager>
```

This code goes in Page_Load():

[C#]

```
DummyDTB.PreLoadForAJAX();
```

[VB]

```
DummyDTB.PreLoadForAJAX()
```

Programmatically:

This code goes in Page_Load():

[C#]

```
PeterBlum.DES.AJAXManager.UsingMicrosoftAJAX();
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
  PeterBlum.DES.AJAXManager.FeatureList.DateTextBoxes);
DummyDTB.PreLoadForAJAX();
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingMicrosoftAJAX()  
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _  
    PeterBlum.DES.AJAXManager.FeatureList.DateTextBoxes )  
DummyDTB.PreLoadForAJAX()
```

Using Telerik RadAjax

<http://www.telerik.com>

ALERT: Peter's Data Entry Suite requires RadAjax version 1.0.1 or higher. Do not use it with RadAjax 1.0.0. It supports Telerik "Prometheus".

RadAJAX supplies two controls that can replace HTML on the page: radAjaxManager and radAjaxPanel. Both are supported by the following steps.

Examples follow these steps:

ALERT: You are virtually guaranteed **JavaScript errors** and **lack of functionality** if you do not implement these steps correctly.

1. For any page with a radAjaxManager or radAjaxPanel control and DES controls, tell DES that you are using RadAJAX.

RadAjax (not the "Prometheus" version)

When using the PageManager:

Set **AJAXFramework** to TelerikRadAJAX and **AJAXControlID** to the ID of the radAjaxManager or radAjaxPanel. Note that these controls must be in the same naming container.

```
<des:PageManager id="PageManager1" runat="server"
    AJAXFramework="TelerikRadAJAX" AJAXControlID="radAjaxControl1" >
</des:PageManager>
```

Programmatically:

Call this method in Page_Load():

```
PeterBlum.DES.AJAXManager.UsingRadAJAX(radajaxmanager1)
PeterBlum.DES.AJAXManager.UsingRadAJAX(radajaxpanel1)
```

ALERT: This method will set the EnableOutsideScripts property to true on the RadAjax control.

RadAjax - The "Prometheus" version

When using the PageManager:

Set **AJAXFramework** to TelerikRadAJAX and **AJAXControlID** to the ID of the Microsoft ASP.NET AJAX ScriptManager control. Note that these controls must be in the same naming container.

```
<des:PageManager id="PageManager1" runat="server"
    AJAXFramework="TelerikRadAJAX" AJAXControlID="ScriptManager1" >
</des:PageManager>
```

Programmatically:

Call this method in Page_Load():

```
PeterBlum.DES.AJAXManager.UsingRadAJAX(ScriptManager.GetCurrent(Page))
```

ALERT: For any code that uses the PeterBlum.DES.AJAXManager, it must run every time Page_Load() is called. Do not nest it inside of an IF statement like `If IsPostBack = False Then`.

2. *Optional step* If you are using the radAjaxManager control, DES can set the **InAJAXUpdate** property to true on all DES controls, avoiding the work described in step 3. There

Add this line to the Application_Start() method of the **Global.asax** file.

[C#]

```
PeterBlum.DES.TelerikWebUI.Globals.AutomaticRadAJAXManager();
```

[VB]

```
PeterBlum.DES.TelerikWebUI.Globals.AutomaticRadAJAXManager()
```

ALERT: Visual Basic users who are using a Web Application Project must prepend the call with their Root Namespace. The Root Namespace is shown in the Project's Properties, under the Assembly tab.

For example: *MyRootNamespace.PeterBlum.DES.TelerikWebUI.Globals.AutomaticRadAJAXManager()*

3. Identify which DES controls are participating in the AJAX update. They need to transfer data to the client-side during each callback. Aside from those inside of an radAjaxPanel and radAjaxManager, include those that point to controls inside of UpdatePanels. Examples: Validators pointing to data entry controls, Enabler conditions, FieldStateControllers, CalculationControllers, and NativeControlExtenders.

The challenge is that if you include controls that are not part of the callback, the callback will have poorer performance because there is more data transferred and processed. To get the best performance, more work is involved in setup.

If you don't include controls that are part of the callback, after the callback those controls either will not operate correctly or will generate javascript errors. So some care is needed.

There are three approaches:

- **Include all controls in the callback**
 - Easy setup, no operating problems after callback.
 - Poorest performance where data transfer is at the maximum. Performs very well when most or all of the DES controls are involved in the callback.
- **Good Performance** – *This is the default setup.*
 - Easy but manual assignment of the **InAJAXUpdate** property is required for controls outside of radAjaxPanels and radAjaxManagers that connect with controls involved in AJAX.
 - Data transfer is minimized although the server does more work.
 - Incorrect setup results in controls malfunctioning.
- **Best Performance**
 - Must identify every control involved in the callback
 - Data transfer is minimized and the server does less work.
 - Incorrect setup results in controls malfunctioning.

They are discussed below.

Include All Controls In the Callback

Choose this approach:

- when most or all DES controls are involved in the Callback
- when you have encountered malfunctions after a callback to see if they go away. (You will then turn it off and set `InAJAXUpdate = true` on the controls with the problems.)

When using the *PageManager* control:

Set the **AllInAJAXUpdate** property to `true`.

```
<des:PageManager runat="server" AJAXFramework="TelerikRadAJAX"
    AJAXControlID="ScriptManager1" AllInAJAXUpdate="True" />
```

Programmatically:

Set the **PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate** property to `true` in `Page_Load()`. This will tell all DES controls to transmit their data during each AJAX callback.

```
PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate = true
```

See “Example 1 – Using the AllInAJAXUpdate property”.

Good Performance

This is the default setup. Performance is determined by how much data is transmitted during the callback and how much work the server has to do to create that data. The best performance is to set the **InAJAXUpdate** property to `true` on every control involved in the callback, as described below in “Best Performance”, below. That requires some work on your part.

This option provides the best performance for data transmitted at the expense of the server performance. The server sets **InAJAXUpdate** after searching for the `radAjaxPanel` as a container control. That searching is where the server does more work. It also sets controls in `radAjaxManager`.

Using this method:

- By default, this method is active. If you have previously set **AllInAJAXUpdate** to `true` (see above) or **SmartSetInAJAXUpdate** to `false`, switch their value. (Those properties are on the *PageManager* control or the **PeterBlum.DES.Globals.Page.AJAXManager** object.)
- For any DES controls outside of `radAjaxPanels` and `radAjaxManager` that point to controls inside of those controls, you must set their **InAJAXUpdate** property to `true`, as shown under “Optimized performance”, below. Examples: *Validators* pointing to data entry controls, *Enabler* conditions, *FieldStateControllers*, *CalculationControllers*, and *NativeControlExtenders*.
- If you did not use the `AutomaticRadAJAXManager()` method described in step 2, add this line to the `Page_Load()` method for each `radAjaxManager` control.

[C#]

```
PeterBlum.DES.TelerikWebUI.Globals.SetUpdatedControlsInAJAXUpdate(radAjaxManager);
```

[VB]

```
PeterBlum.DES.TelerikWebUI.Globals.SetUpdatedControlsInAJAXUpdate(radAjaxManager)
```

ALERT: Visual Basic users who are using a Web Application Project must prepend the call with their Root Namespace. The Root Namespace is shown in the Project’s Properties, under the Assembly tab.

For example: *MyRootNamespace.PeterBlum.DES.TelerikWebUI.Globals.SetUpdatedControlsInAJAXUpdate()*

- Feel free to set **InAJAXUpdate** to `true` on any DES controls that are part of the callback to avoid having the CPU search for the `UpdatePanel`. For any you don’t set, DES will do the detection of the `UpdatePanel`.

Best Performance

By setting **InAJAXUpdate** to `true` on exactly those DES controls involved in the callback, transmission time and CPU time are reduced.

Using this method:

- Set to `false` the two properties that make setup easier, but less optimal: **AllInAJAXUpdate** and **SmartSetInAJAXUpdate**. **AllInAJAXUpdate** defaults to `false`, so you probably won't need to do anything to it.

When using the *PageManager* control:

Set the **SmartSetInAJAXUpdate** property to `false`.

```
<des:PageManager runat="server" AJAXFramework="TelerikRadAJAX"
  AJAXControlID="ScriptManager1" SmartSetInAJAXUpdate="False" />
```

Programmatically:

Set the **PeterBlum.DES.AJAXManager.Current.SmartSetInAJAXUpdate** property to `false` in `Page_Load()`. This will tell all DES controls to transmit their data during each AJAX callback.

```
PeterBlum.DES.AJAXManager.Current.SmartSetInAJAXUpdate = false
```

- Every DES control has the **InAJAXUpdate** property. Set it to `true` on those that are in the `radAjaxPanel` and `radAjaxManager`, or are connected to controls inside them. For example:

```
<des:RequiredTextValidator InAJAXUpdate="true" [other properties] />
<des:TextBox InAJAXUpdate="true" [other properties] />
<des:FieldStateController InAJAXUpdate="true" [other properties] />
<des:Button InAJAXUpdate="true" [other properties] />
<des:CalculationController InAJAXUpdate="true" [other properties] />
```

- To analyze how the **InAJAXUpdate** property on DES's controls have been set up, see "Analyzing the InAJAXUpdate Properties on DES Controls".

See "Example 2 – Identify individual DES Controls for Best Performance".

Visual Studio and Visual Web Developer design mode suggestions:

- Select all DES controls that are in the AJAX update. The Properties Editor will display all properties they have in common, including **InAJAXUpdate**. Set it to `true` to update all at once.
- ASP.NET 2 only.* Each DES control has a SmartTag with the setting **Involved in an AJAX Update**. Mark its checkbox.

To let the computer do some of the work, use the

`PeterBlum.DES.AJAXManager.SetChildrenInAJAXUpdate()` method to update a container control, like a panel, so that all of its child DES controls have their **InAJAXUpdate** property set to `true`. If any DES control points to another control outside of the container, its **InAJAXUpdate** property will also be set to `true`. However, if there is a DES control outside the container that is not pointed to by a control inside the container, it will still need you to set **InAJAXUpdate**. See "PeterBlum.DES.AJAXManager.SetChildrenInAJAXUpdate".

CONTINUE TO THE NEXT PAGE FOR STEPS 4 AND 5

4. If a type of DES control or feature is not initially added to page but will later be added during a callback, it needs to be identified when the page is initially requested (`Page.IsPostBack = false`). Examples include a `DataGrid` or `GridView` that switches to edit mode with textboxes and validators; and the `Wizard` which often does not have data entry controls on the first step.

When using the `PageManager` control:

Using the chart on the next page, set the desired property to true on the **`PageManager.PreLoadForAJAX`** property. For example:

```
<des:PageManager runat="server" PreLoadForAJAX-Validators="True"
PreLoadForAJAX-TextBoxes="True" />
```

Programmatically:

Call `PeterBlum.DES.AJAXManager.PreregisterForAJAX()`.

You pass the enumerated type `PeterBlum.DES.AJAXManager.FeatureList`, whose values are listed in the chart below. For example:

```
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.Validators)
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.TextBoxes)
```

Note: It is safe to call `PreregisterForAJAX()` even when the controls are shown when the page is first generated. All it does is guarantee scripts are added to the page. If you call it when the controls will never be shown, it adds the overhead of loading script files and setting up small global objects.

The `PreregisterForAJAX()` method also accepts an array of `FeatureList` values like this:

[C#]

```
using PeterBlum.DES;
...
AJAXManager.FeatureList[] vFeatures =
    {AJAXManager.FeatureList.Validators, AJAXManager.FeatureList.TextBoxes};
AJAXManager.PreregisterForAJAX(vFeatures);
```

[VB]

```
Imports PeterBlum.DES
...
Dim vFeatures() As AJAXManager.FeatureList = New AJAXManager.FeatureList() _
    {AJAXManager.FeatureList.Validators, AJAXManager.FeatureList.TextBoxes}
AJAXManager.PreregisterForAJAX(vFeatures)
```

See “Example 3 – Controls added during Callback”.

CHART OF FEATURES IS ON THE NEXT PAGE

AJAXManager.FeatureList values and PageManager.PreLoadForAJAX properties	
Validators	Use when no validator is on the initial page but will be added during a callback.
ValidationSummary	Use when no ValidationSummary is on the initial page but will be added during a callback.
CombinedErrorMessages	Use when a CombinedErrorMessages control is added on a callback.
TextBoxes	Use when no DES TextBox is on the initial page but will be added during a callback. This includes TextBox, FilteredTextBox, IntegerTextBox, DecimalTextBox, PercentTextBox, and CurrencyTextBox. Also consider Hints, AutoPostBack.
MultiSegmentDataEntry	Use when no MultiSegmentDataEntry control is on the initial page but will be added during a callback. Also consider Hints.
Hints (Also when using ToolTips)	Use when no hints are on the initial page but will be used during a callback. This also applies when using the hint properties on DES's textboxes and MultiSegmentDataEntry controls. By default, ToolTips are converted to hints. When using ToolTips, you need to preload the Hints feature or to turn off the conversion of ToolTips to hints with the ToolTipsAsHints property on the PageManager control or PeterBlum.DES.Globals.Page . See "Properties on the PeterBlum.DES.Globals.Page.HintManager Property" topic in the Interactive Pages Guide .
DateTextBoxes *	Use when no DateTextBox, AnniversaryTextBox, or MonthYearTextBox is on the initial page but will be added during the callback. Also consider Hints, AutoPostBack.
TimeTextBoxes *	Use when no TimeOfDayTextBox or DurationTextBox is on the initial page but will be added during the callback. Also consider Hints, AutoPostBack.
Calendar *	Use when no Calendar is on the initial page but will be added during the callback.
MultiSelectionCalendar *	Use when no MultiSelectionCalendar is on the initial page but will be added during the callback.
MonthYearPicker *	Use when no MonthYearPicker or PopupMonthYearPicker is on the initial page but will be added during the callback.
TimePicker *	Use when no TimePicker or PopupTimePicker is on the initial page but will be added during the callback.
SpecialDates	Use when no SpecialDates control is on the initial page but will be added during the callback.
ContextMenu	Use when no ContextMenu control is on the initial page but will be added during the callback.
FieldStateControllers	Use when no type of FieldStateController is on the initial page but will be added during a callback. This includes FieldStateController, MultiFieldStateController, FSCOnCommand, and MultiFSCOnCommand.
CalculationControllers	Use when no CalculationController is on the initial page but will be added during a callback.
TextCounter	Use when no TextCounter is on the initial page but will be added during the callback.
AutoPostBack	Use when you validate on AutoPostBack.
ChangeMonitor	Use when the ChangeMonitor is added to the page after postback.
SubmitControls	Use when no Button, LinkButton, or ImageButton is on the initial page but will be added during the callback.
DisableOnSubmit	Use when no Button uses the DisableOnSubmit feature on the initial page but will be used during a callback.
Popups	Use when you create a control using the Popup code.

* The next step is required for these controls.

5. For any of the following items from the above chart: DateTextBoxes, TimeTextBoxes, Calendar, MultiSelectionCalendar, MonthYearPicker, and TimePicker, their popup features must be preloaded during the initial page load, even though the control itself is not shown until the AJAX update.

Follow these instructions for *each type of control* that was identified:

- Create a new instance of the control to the page. Put it just after the <form> tag in the web form (aspx) file or at the beginning of the UserControl. *It should **not** be inside the UpdatePanel or radAJAXPanel.* This will avoid having it enclosed in another web control that may have its **Visible** property set to false.

```
<form id="form1" runat="server" >
  <des:DateTextBox id="DummyDTB" runat="server" />
  <asp:othercontrols here />
```

Make sure this control is always added to the page, even if you are not sure if same type of control will actually be added during a callback. **WARNING:** *It must be added to the Page's Control tree before any of the same type controls that you plan to show in the radAJAX panel because its PreRender method must run first. Otherwise, you will get JavaScript errors after the callback.*

- Set up the popup elements on this control the way you want to see them when the control is shown in the browser. These popups will be written out when the page is first generated.

```
<des:DateTextBox id="DummyDTB" runat="server" >
  <PopupCalendar>
    <Calendar propertyname="value" propertyname2="value" />
  </PopupCalendar>
</des:DateTextBox>
```

- Call the new control's PreLoadForAJAX() method in Page_Load().

```
DummyDTB.PreLoadForAJAX( )
```

The call to PreLoadForAJAX() prevents this control from appearing on the page, but DES will use it to preload the HTML and scripts of its popups.

See "Example 4 – Control with Popups first created during Callback".

Example 2 – Identify individual DES Controls for Best Performance

Suppose you have a radAjaxManager control that updates a DES TextBox and an associated RequiredTextValidator when a DES Button is clicked. You want to use the **InAJAXUpdate** property on each DES control in the UpdatePanel because there are other DES controls on the page.

*Note: To analyze how the **InAJAXUpdate** property on DES's controls have been set up, see "Analyzing the InAJAXUpdate Properties on DES Controls".*

```
<radA:RadAjaxManager id="RadAjaxManager1" runat="server">
<AjaxSettings>
  <rada:AjaxSetting AjaxControlID="Button1">
    <UpdatedControls>
      <rada:AjaxUpdatedControl ControlID="TextBox1"></rada:AjaxUpdatedControl>
      <rada:AjaxUpdatedControl ControlID="RTV1"></rada:AjaxUpdatedControl>
    </UpdatedControls>
  </rada:AjaxSetting>
</AjaxSettings>
</radA:RadAjaxManager>

<des:TextBox id="TextBox1" runat="server" InAJAXUpdate="true" />&nbsp;
<des:RequiredTextValidator id="RTV1" runat="server" InAJAXUpdate="true"
  ControlIDToEvaluate="TextBox1" ErrorMessage="Required" /><br />
<des:Button id="Submit" runat="server" Text="Submit" InAJAXUpdate="true" />
```

Using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
  AJAXFramework="TelerikRadAJAX" AJAXControlID="RadAjaxManager1">
</des:PageManager>
```

Programmatically:

This code goes in Page_Load():

[C#]

```
PeterBlum.DES.AJAXManager.UsingRadAJAX(RadAjaxManager1);
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingRadAJAX(RadAjaxManager1)
```

Example 3 – Controls added during Callback

Suppose you have a radAjaxManager control that updates a DES TextBox and an associated RequiredTextValidator when a DES Button is clicked. The TextBox and Validator controls are inside of a Panel which is **Visible=false** until the DES Button sets it to **Visible=true** during a callback.

```
<radA:RadAjaxManager id="RadAjaxManager1" runat="server">
  <AjaxSettings>
    <rada:AjaxSetting AjaxControlID="Button1">
      <UpdatedControls>
        <rada:AjaxUpdatedControl ControlID="TextBox1"></rada:AjaxUpdatedControl>
        <rada:AjaxUpdatedControl ControlID="RTV1"></rada:AjaxUpdatedControl>
      </UpdatedControls>
    </rada:AjaxSetting>
  </AjaxSettings>
</radA:RadAjaxManager>

<asp:Panel id="Panell" runat="server" Visible="false">
  <des:TextBox id="TextBox1" runat="server" InAJAXUpdate="true" />&nbsp;
  <des:RequiredTextValidator id="RTV1" runat="server" InAJAXUpdate="true"
    ControlIDToEvaluate="TextBox1" ErrorMessage="Required" /><br />
</asp:Panel>
<des:Button id="Show" runat="server" Text="Submit" InAJAXUpdate="true"
  Click="Show_ButtonClick" />
```

Using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
  AJAXFramework="TelerikRadAJAX" AJAXControlID="RadAjaxManager1"
  PreLoadForAJAX-Validators="True" PreLoadForAJAX-TextBoxes="True">
</des:PageManager>
```

Programmatically:

This code goes in Page_Load():

[C#]

```
PeterBlum.DES.AJAXManager.UsingRadAJAX(RadAjaxManager1);
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
  PeterBlum.DES.AJAXManager.FeatureList.Validators);
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
  PeterBlum.DES.AJAXManager.FeatureList.TextBoxes);
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingRadAJAX(RadAjaxManager1)
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _
  PeterBlum.DES.AJAXManager.FeatureList.Validators)
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _
  PeterBlum.DES.AJAXManager.FeatureList.TextBoxes)
```

This code goes in Show_ButtonClick():

[C#]

```
Panell.Visible = true;
```

[VB]

```
Panell.Visible = True
```

Example 4 – Control with Popups first created during Callback

Suppose you have a RadAjaxManager control with GridView that shows a DateTextBox in edit mode. Until the GridView is in edit mode, the <EditItemTemplate> is not created, requiring a dummy DateTextBox to preload the popups. The dummy DateTextBox's popups must have the same properties as the actual DateTextBox. In this case, **CssClass** is set to a custom style sheet class. *Reminder: The Dummy control must not be updated by the RadAjaxManager or RadAjaxPanel. Try to get it just below the <form> tag.*

```
<des:DateTextBox id="DummyDTB" runat="server" >
  <PopupCalendar>
    <Calendar CssClass="MyCalendarClass" />
  </PopupCalendar>
</des:DateTextBox>

<radA:RadAjaxManager id="RadAjaxManager1" runat="server">
<AjaxSettings>
  <rada:AjaxSetting AjaxControlID="GridView1">
    <UpdatedControls>
      <rada:AjaxUpdatedControl ControlID="GridView1"></rada:AjaxUpdatedControl>
    </UpdatedControls>
  </rada:AjaxSetting>
</AjaxSettings>
</radA:RadAjaxManager>

<asp:GridView id="GridView1" runat="server" OnRowCreated="GridView1_RowCreated" >
<Columns>
  <des:CommandField ShowEditButton="True" />
  <asp:TemplateField>
    <ItemTemplate>...</ItemTemplate>
    <EditItemTemplate>
      <des:DateTextBox id="DateTextBox1" runat="server" InAJAXUpdate="true" >
        <PopupCalendar>
          <Calendar CssClass="MyCalendarClass" />
        </PopupCalendar>
      </des:DateTextBox>
    </EditItemTemplate>
  </asp:TemplateField>
</Columns>
</asp:GridView>
```

Using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
  AJAXFramework="TelerikRadAJAX" AJAXControlID="RadAjaxManager1"
  PreLoadForAJAX-DateTextBoxes="True" >
</des:PageManager>
```

This code goes in Page_Load ():

[C#]

```
DummyDTB.PreLoadForAJAX( );
```

[VB]

```
DummyDTB.PreLoadForAJAX( )
```

Programmatically:

This code goes in Page_Load ():

[C#]

```
PeterBlum.DES.AJAXManager.UsingRadAJAX(RadAjaxManager1);  
PeterBlum.DES.AJAXManager.PreregisterForAJAX(  
    PeterBlum.DES.AJAXManager.FeatureList.DateTextBoxes);  
DummyDTB.PreLoadForAJAX();
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingRadAJAX(RadAjaxManager1)  
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _  
    PeterBlum.DES.AJAXManager.FeatureList.DateTextBoxes)  
DummyDTB.PreLoadForAJAX()
```

Using other AJAX-enabled Telerik controls

<http://www.telerik.com>

Telerik's RadControls are AJAX enabled when they offer the property **EnableAJAX**.

ALERT: You are virtually guaranteed **JavaScript errors** and **lack of functionality** if you do not implement these steps correctly.

- For each AJAX-enabled radControl that *contains* any DES controls or controls pointed to by a DES control, call this method in Page_Load():


```
PeterBlum.DES.AJAXManager.UsingAJAXUpdates(Page,
      PeterBlum.DES.AJAXManager.AJAXSystemType.RegisterScripts,
      control.IsAjaxRequest)
```
- Set the property **EnableOutsideScripts** to true on the radControl.
- Identify which DES controls are participating in the AJAX update. Aside from those inside of an AJAX-enabled RadControl, include those that point to controls involved in AJAX updates. Examples: Validators pointing to data entry controls, Enabler conditions, FieldStateControllers, CalculationControllers, and NativeControlExtenders.

There are two approaches:

- Quick Setup** – Use a single setting to establish that every DES control on the page as involved in the AJAX update (even if its not). It provides a fast setup but will make AJAX updates perform less than optimally because all DES controls will transmit their data.
- Optimized Performance** – Identify individual controls involved in the AJAX update. Allows only those controls to transmit their HTML and JavaScript. This can greatly improve performance during a callback.

Recommendation: Use Quick Setup during initial page design. If most or all DES controls are involved in an AJAX update, retain this set up. Otherwise, switch to the Optimized performance setup.

They are both discussed below.

Quick Setup

When using the PageManager control:

Set the **AllInAJAXUpdate** property to true.

```
<des:PageManager runat="server" AllInAJAXUpdate="True" />
```

Programmatically:

Set the **PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate** property to true in Page_Load(). This will tell all DES controls to transmit their data during each AJAX callback.

```
PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate = true
```

See “Example 1 – Using the AllInAJAXUpdate property for “Quick Setup””.

Optimized performance

Every DES control has the **InAJAXUpdate** property. Set it to true. For example:

```
<des:RequiredTextValidator InAJAXUpdate="true" [other properties] />
```

```
<des:TextBox InAJAXUpdate="true" [other properties] />
```

```
<des:FieldStateController InAJAXUpdate="true" [other properties] />
```

```
<des:Button InAJAXUpdate="true" [other properties] />
```

```
<des:CalculationController InAJAXUpdate="true" [other properties] />
```

Note: To analyze how the **InAJAXUpdate** property on DES's controls have been set up, see “Analyzing the InAJAXUpdate Properties on DES Controls”.

See “Example 2 – Identify individual DES Controls for Optimized Performance”.

Visual Studio and Visual Web Developer design mode suggestions:

- Select all DES controls that are in the AJAX update. The Properties Editor will display all properties they have in common, including **InAJAXUpdate**. Set it to `true` to update all at once.
- *ASP.NET 2 only.* Each DES control has a SmartTag with the setting **Involved in an AJAX Update**. Mark its checkbox.

To let the computer do some of the work, use the

`PeterBlum.DES.AJAXManager.SetChildrenInAJAXUpdate()` method to update a container control, like a panel, so that all of its child DES controls have their **InAJAXUpdate** property set to `true`. If any DES control points to another control outside of the container, its **InAJAXUpdate** property will also be set to `true`. However, if there is a DES control outside the container that is not pointed to by a control inside the container, it will still need you to set **InAJAXUpdate**. See “`PeterBlum.DES.AJAXManager.SetChildrenInAJAXUpdate`”.

To analyze how the **InAJAXUpdate** property on DES’s controls have been set up, simply set `<%@ Page Trace="True" %>`. It will output a list of all relevant DES controls on the page with the value of their **InAJAXUpdate** property.

4. If a type of DES control or feature is not initially added to page but will later be added during a callback, it needs to be identified when the page is initially requested (`Page.IsPostBack = false`). Examples include a `DataGrid` or `GridView` that switches to edit mode with textboxes and validators; and the `Wizard` which often does not have data entry controls on the first step.

When using the PageManager control:

Using the chart below, set the desired property to true on the **PageManager.PreLoadForAJAX** property. For example:

```
<des:PageManager runat="server" PreLoadForAJAX-Validators="True"
PreLoadForAJAX-TextBoxes="True" />
```

Programmatically:

Call `PeterBlum.DES.AJAXManager.PreregisterForAJAX()`.

You pass the enumerated type `PeterBlum.DES.AJAXManager.FeatureList`, whose values are listed in the chart below. For example:

```
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.Validators)
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.TextBoxes)
```

Note: It is safe to call `PreregisterForAJAX()` even when the controls are shown when the page is first generated. All it does is guarantee scripts are added to the page. If you call it when the controls will never be shown, it adds the overhead of loading script files and setting up small global objects.

The `PreregisterForAJAX()` method also accepts an array of `FeatureList` values like this:

[C#]

```
using PeterBlum.DES;
...
AJAXManager.FeatureList[] vFeatures =
    {AJAXManager.FeatureList.Validators, AJAXManager.FeatureList.TextBoxes};
AJAXManager.PreregisterForAJAX(vFeatures);
```

[VB]

```
Imports PeterBlum.DES
...
Dim vFeatures() As AJAXManager.FeatureList = New AJAXManager.FeatureList() _
    {AJAXManager.FeatureList.Validators, AJAXManager.FeatureList.TextBoxes}
AJAXManager.PreregisterForAJAX(vFeatures)
```

See "Example 3 – Controls added during Callback".

AJAXManager.FeatureList values and PageManager.PreLoadForAJAX properties	
Validators	Use when no validator is on the initial page but will be added during a callback.
ValidationSummary	Use when no ValidationSummary is on the initial page but will be added during a callback.
CombinedErrorMessages	Use when a CombinedErrorMessages control is added on a callback.
TextBoxes	Use when no DES TextBox is on the initial page but will be added during a callback. This includes TextBox, FilteredTextBox, IntegerTextBox, DecimalTextBox, PercentTextBox, and CurrencyTextBox. Also consider Hints, AutoPostBack.
MultiSegmentDataEntry	Use when no MultiSegmentDataEntry control is on the initial page but will be added during a callback. Also consider Hints.
Hints (Also when using ToolTips)	Use when no hints are on the initial page but will be used during a callback. This also applies when using the hint properties on DES's textboxes and MultiSegmentDataEntry controls. By default, ToolTips are converted to hints. When using ToolTips, you need to preload the Hints feature or to turn off the conversion of ToolTips to hints with the ToolTipsAsHints property on the PageManager control or PeterBlum.DES.Globals.Page .
DateTextBoxes *	Use when no DateTextBox, AnniversaryTextBox, or MonthYearTextBox is on the initial page but will be added during the callback. Also consider Hints, AutoPostBack.
TimeTextBoxes *	Use when no TimeOfDayTextBox or DurationTextBox is on the initial page but will be added during the callback. Also consider Hints, AutoPostBack.
Calendar *	Use when no Calendar is on the initial page but will be added during the callback.
MultiSelectionCalendar*	Use when no MultiSelectionCalendar is on the initial page but will be added during the callback.
MonthYearPicker *	Use when no MonthYearPicker or PopupMonthYearPicker is on the initial page but will be added during the callback.
TimePicker *	Use when no TimePicker or PopupTimePicker is on the initial page but will be added during the callback.
SpecialDates	Use when no SpecialDates control is on the initial page but will be added during the callback.
ContextMenu	Use when no ContextMenu control is on the initial page but will be added during the callback.
FieldStateControllers	Use when no type of FieldStateController is on the initial page but will be added during a callback. This includes FieldStateController, MultiFieldStateController, FSCOnCommand, and MultiFSCOnCommand.
CalculationControllers	Use when no CalculationController is on the initial page but will be added during a callback.
TextCounter	Use when no TextCounter is on the initial page but will be added during the callback.
AutoPostBack	Use when you validate on AutoPostBack.
ChangeMonitor	Use when the ChangeMonitor is added to the page after postback.
SubmitControls	Use when no Button, LinkButton, or ImageButton is on the initial page but will be added during the callback.
DisableOnSubmit	Use when no Button uses the DisableOnSubmit feature on the initial page but will be used during a callback.
Popups	Use when you create a control using the Popup code.

* The next step is required for these controls.

5. For any of the following items from the above chart: DateTextBoxes, TimeTextBoxes, Calendar, MultiSelectionCalendar, MonthYearPicker, and TimePicker, their popup features must be preloaded during the initial page load, even though the control itself is not shown until the AJAX update.

Follow these instructions for *each type of control* that was identified:

- Create a new instance of the control to the page. Put it just after the <form> tag in the web form (aspx) file or at the beginning of the UserControl. This will avoid having it enclosed in another web control that may have its **Visible** property set to false.

```
<form id="form1" runat="server" >
  <des:DateTextBox id="DummyDTB" runat="server" />
  <asp:othercontrols here />
```

Make sure this control is always added to the page, even if you are not sure if same type of control will actually be added during a callback. **WARNING:** *It must be added to the Page's Control tree before any of the same type controls that you plan to show in the AJAX panel because its PreRender method must run first. Otherwise, you will get JavaScript errors after the callback.*

- Set up the popup elements on this control the way you want to see them when the control is shown in the browser. These popups will be written out when the page is first generated.

```
<des:DateTextBox id="DummyDTB" runat="server" >
  <PopupCalendar>
    <Calendar propertyname="value" propertyname2="value" />
  </PopupCalendar>
</des:DateTextBox>
```

- Call the new control's PreLoadForAJAX() method in Page_Load().

```
DummyDTB.PreLoadForAJAX( )
```

The call to PreLoadForAJAX() prevents this control from appearing on the page, but DES will use it to preload the HTML and scripts of its popups.

See "Example 4 – Control with Popups first created during Callback".

Example 2 – Identify individual DES Controls for Optimized Performance

Suppose you have a radGrid with a DES TextBox and an associated RequiredTextValidator in a GridTemplateColumn. You want to use the **InAJAXUpdate** property on each DES control in the radGrid because there are other DES controls on the page.

*Note: To analyze how the **InAJAXUpdate** property on DES's controls have been set up, see "Analyzing the InAJAXUpdate Properties on DES Controls".*

```
<radg:GridTemplateColumn [other properties]>
    <des:TextBox id="TextBox1" runat="server" InAJAXUpdate="true" />&nbsp;
    <des:RequiredTextValidator id="RTV1" runat="server" InAJAXUpdate="true"
    ControlIDToEvaluate="TextBox1" ErrorMessage="Required" /><br />
</radg:GridTemplateColumn>
```

This code goes in Page_Load():

[C#]

```
PeterBlum.DES.AJAXManager.UsingAJAXUpdates(Page,
    PeterBlum.DES.AJAXManager.AJAXSystemType.RegisterScripts,
    radGrid1.IsAjaxRequest);
radGrid1.EnableOutsideScripts = true;
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingAJAXUpdates(Page, _
    PeterBlum.DES.AJAXManager.AJAXSystemType.RegisterScripts, _
    radGrid1.IsAjaxRequest)
radGrid1.EnableOutsideScripts = True
```

Example 3 – Controls added during Callback

Suppose you have a radGrid with a DES TextBox and an associated RequiredTextValidator in a GridTemplateColumn. The TextBox and Validator controls are not shown until the user invokes Edit mode on the grid, which requires a callback to get the editing controls.

```
<radg:GridTemplateColumn [other properties]>
    <des:TextBox id="TextBox1" runat="server" InAJAXUpdate="true" />&nbsp;
    <des:RequiredTextValidator id="RTV1" runat="server" InAJAXUpdate="true"
        ControlIDToEvaluate="TextBox1" ErrorMessage="Required" /><br />
</radg:GridTemplateColumn>
```

This code goes in Page_Load():

[C#]

```
PeterBlum.DES.AJAXManager.UsingAJAXUpdates(Page,
    PeterBlum.DES.AJAXManager.AJAXSystemType.RegisterScripts,
    radGrid1.IsAjaxRequest);
radGrid1.EnableOutsideScripts = true;
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.Validators);
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.TextBoxes);
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingAJAXUpdates(Page, _
    PeterBlum.DES.AJAXManager.AJAXSystemType.RegisterScripts, _
    radGrid1.IsAjaxRequest)
radGrid1.EnableOutsideScripts = True
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _
    PeterBlum.DES.AJAXManager.FeatureList.Validators)
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _
    PeterBlum.DES.AJAXManager.FeatureList.TextBoxes)
```

Example 4 – Control with Popups first created during Callback

Suppose you have radGrid that shows a DateTextBox in edit mode. Until the radGrid is in edit mode, the <EditItemTemplate> is not created, requiring a dummy DateTextBox to preload the popups. The dummy DateTextBox's popups must have the same properties as the actual DateTextBox. In this case, **CssClass** is set to a custom style sheet class.

```
<des:DateTextBox id="DummyDTB" runat="server" >
  <PopupCalendar>
    <Calendar CssClass="MyCalendarClass" />
  </PopupCalendar>
</des:DateTextBox>

<radg:RadGrid id="RadGrid1" runat="server" [other properties]>
<Columns>
  <vrg:GridEditCommandColumn [other properties]/>
  <radg:GridTemplateColumn [other properties]>
    <ItemTemplate>...</ItemTemplate>
    <EditItemTemplate>

      <des:DateTextBox id="DateTextBox1" runat="server" InAJAXUpdate="true" >
        <PopupCalendar>
          <Calendar CssClass="MyCalendarClass" />
        </PopupCalendar>
      </des:DateTextBox>

    </EditItemTemplate>
  </radg:GridTemplateColumn>
</Columns>
</asp:GridView>
```

Using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
  PreLoadForAJAX-DateTextBoxes="True" >
</des:PageManager>
```

This code goes in Page_Load ():

[C#]

```
PeterBlum.DES.AJAXManager.UsingAJAXUpdates(Page,
  PeterBlum.DES.AJAXManager.AJAXSystemType.RegisterScripts,
  RadGrid1.IsAjaxRequest);
DummyDTB.PreLoadForAJAX();
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingAJAXUpdates(Page, _
  PeterBlum.DES.AJAXManager.AJAXSystemType.RegisterScripts, _
  RadGrid1.IsAjaxRequest)
DummyDTB.PreLoadForAJAX();
```

Programmatically:

This code goes in Page_Load ():

[C#]

```
PeterBlum.DES.AJAXManager.UsingAJAXUpdates(Page,
  PeterBlum.DES.AJAXManager.AJAXSystemType.RegisterScripts,
  RadGrid1.IsAjaxRequest);
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
  PeterBlum.DES.AJAXManager.FeatureList.DateTextBoxes);
DummyDTB.PreLoadForAJAX();
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingAJAXUpdates(Page, _  
    PeterBlum.DES.AJAXManager.AJAXSystemType.RegisterScripts, _  
    RadGrid1.IsAjaxRequest)  
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _  
    PeterBlum.DES.AJAXManager.FeatureList.DateTextBoxes)  
DummyDTB.PreLoadForAJAX( )
```

Using Infragistics AJAX-enabled Controls

<http://www.infragistics.com>

ALERT: Peter's Data Entry Suite was tested with Infragistics NetAdvantage 2006 Q3.

NetAdvantage supplies two controls that can replace HTML: WebAsyncRefreshPanel and UltraWebTab. Both are supported by the following steps. In addition, any control introduced by Infragistics that offers the **IsAsyncPostBack** property is anticipated to work.

ALERT: You are virtually guaranteed **JavaScript errors and lack of functionality** if you do not implement these steps correctly.

1. For any page with a WebAsyncRefreshPanel or UltraWebTab that contains or is in some way connected to a DES control, identify it.

When using the PageManager:

Set **AJAXFramework** to InfragisticsAJAX and **AJAXControlID** to the ID of the WebAsyncRefreshPanel or UltraWebTab. Note that these controls must be in the same naming container. Note: This only supports a single Infragistics control. If you have more, use the programmatic method below.

```
<des:PageManager runat="server" AJAXFramework="InfragisticsAJAX"
AJAXControlID="WebAsyncRefreshPanel" />
```

Programmatically:

Call this method in Page_Load() for each control:

```
PeterBlum.DES.AJAXManager.UsingInfragisticsAJAX(WebAsyncRefreshPanel)
PeterBlum.DES.AJAXManager.UsingInfragisticsAJAX(UltraWebTag)
```

ALERT: For any code that uses the PeterBlum.DES.AJAXManager, it must run every time Page_Load() is called. Do not nest it inside of an IF statement like If IsPostBack = False Then.

2. Identify which DES controls are participating in the AJAX update. Aside from those inside of an AJAX-enabled control, include those that point to controls involved in an AJAX update. Examples: Validators pointing to data entry controls, Enabler conditions, FieldStateControllers, CalculationControllers, and NativeControlExtenders.

There are two approaches:

- Quick Setup – Use a single setting to establish that every DES control on the page as involved in the AJAX update (even if its not). It provides a fast setup but will make AJAX updates perform less than optimally because all DES controls will transmit their data.
- Optimized Performance – Identify individual controls involved in the AJAX update. Allows only those controls to transmit their HTML and JavaScript. This can greatly improve performance during a callback.

Recommendation: Use Quick Setup during initial page design. If most or all DES controls are involved in an AJAX update, retain this set up. Otherwise, switch to the Optimized performance setup.

They are both discussed below.

Quick Setup

When using the PageManager control:

Set the **AllInAJAXUpdate** property to true.

```
<des:PageManager runat="server"
AJAXFramework="InfragisticsAJAX" AllInAJAXUpdate="True" />
```

See “Example 1 – Using the AllInAJAXUpdate property for “Quick Setup””.

Programmatically:

Set the **PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate** property to true in Page_Load(). This will tell all DES controls to transmit their data during each AJAX callback.

```
PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate = true
```

Optimized performance

Every DES control has the **InAJAXUpdate** property. Set it to true. For example:

```
<des:RequiredTextValidator InAJAXUpdate="true" [other properties] />
<des:TextBox InAJAXUpdate="true" [other properties] />
<des:FieldStateController InAJAXUpdate="true" [other properties] />
<des:Button InAJAXUpdate="true" [other properties] />
<des:CalculationController InAJAXUpdate="true" [other properties] />
```

*Note: To analyze how the **InAJAXUpdate** property on DES's controls have been set up, see "Analyzing the InAJAXUpdate Properties on DES Controls".*

See "Example 2 – Identify individual DES Controls for Optimized Performance".

Visual Studio and Visual Web Developer design mode suggestions:

- Select all DES controls that are in the AJAX update. The Properties Editor will display all properties they have in common, including **InAJAXUpdate**. Set it to true to update all at once.
- *ASP.NET 2 only.* Each DES control has a SmartTag with the setting **Involved in an AJAX Update**. Mark its checkbox.

To let the computer do some of the work, use the

`PeterBlum.DES.AJAXManager.SetChildrenInAJAXUpdate()` method to update a container control, like a panel, so that all of its child DES controls have their **InAJAXUpdate** property set to true. If any DES control points to another control outside of the container, its **InAJAXUpdate** property will also be set to true. However, if there is a DES control outside the container that is not pointed to by a control inside the container, it will still need you to set **InAJAXUpdate**. See "PeterBlum.DES.AJAXManager.SetChildrenInAJAXUpdate".

3. If a type of DES control or feature is not initially added to page but will later be added during a callback, it needs to be identified when the page is initially requested (`Page.IsPostBack = false`). Examples include a DataGrid or GridView that switches to edit mode with textboxes and validators; and the Wizard which often does not have data entry controls on the first step.

When using the PageManager control:

Using the chart below, set the desired property to true on the **PageManager.PreLoadForAJAX** property. For example:

```
<des:PageManager runat="server" PreLoadForAJAX-Validators="True"
PreLoadForAJAX-TextBoxes="True" />
```

Programmatically:

Call `PeterBlum.DES.AJAXManager.PreregisterForAJAX()`.

You pass the enumerated type `PeterBlum.DES.AJAXManager.FeatureList`, whose values are listed in the chart below. For example:

```
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.Validators)
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.TextBoxes)
```

Note: It is safe to call `PreregisterForAJAX()` even when the controls are shown when the page is first generated. All it does is guarantee scripts are added to the page. If you call it when the controls will never be shown, it adds the overhead of loading script files and setting up small global objects.

The PreregisterForAJAX() method also accepts an array of FeatureList values like this:

[C#]

```
using PeterBlum.DES;
...
AJAXManager.FeatureList[] vFeatures =
    {AJAXManager.FeatureList.Validators, AJAXManager.FeatureList.TextBoxes};
AJAXManager.PreregisterForAJAX(vFeatures);
```

[VB]

```
Imports PeterBlum.DES
...
Dim vFeatures() As AJAXManager.FeatureList = New AJAXManager.FeatureList() _
    {AJAXManager.FeatureList.Validators, AJAXManager.FeatureList.TextBoxes}
AJAXManager.PreregisterForAJAX(vFeatures)
```

See “Example 3 – Controls added during Callback”.

AJAXManager.FeatureList values and PageManager.PreLoadForAJAX properties	
Validators	Use when no validator is on the initial page but will be added during a callback.
ValidationSummary	Use when no ValidationSummary is on the initial page but will be added during a callback.
CombinedErrorMessages	Use when a CombinedErrorMessages control is added on a callback.
TextBoxes	Use when no DES TextBox is on the initial page but will be added during a callback. This includes TextBox, FilteredTextBox, IntegerTextBox, DecimalTextBox, PercentTextBox, and CurrencyTextBox. Also consider Hints, AutoPostBack.
MultiSegmentDataEntry	Use when no MultiSegmentDataEntry control is on the initial page but will be added during a callback. Also consider Hints.
Hints (Also when using ToolTips)	Use when no hints are on the initial page but will be used during a callback. This also applies when using the hint properties on DES’s textboxes and MultiSegmentDataEntry controls. By default, ToolTips are converted to hints. When using ToolTips, you need to preload the Hints feature or to turn off the conversion of ToolTips to hints with the ToolTipsAsHints property on the PageManager control or PeterBlum.DES.Globals.Page . See “Properties on the PeterBlum.DES.Globals.Page.HintManager Property” topic in the Interactive Pages Guide .
DateTextBoxes *	Use when no DateTextBox, AnniversaryTextBox, or MonthYearTextBox is on the initial page but will be added during the callback. Also consider Hints, AutoPostBack.
TimeTextBoxes *	Use when no TimeOfDayTextBox or DurationTextBox is on the initial page but will be added during the callback. Also consider Hints, AutoPostBack.
Calendar *	Use when no Calendar is on the initial page but will be added during the callback.
MultiSelectionCalendar*	Use when no MultiSelectionCalendar is on the initial page but will be added during the callback.
MonthYearPicker *	Use when no MonthYearPicker or PopupMonthYearPicker is on the initial page but will be added during the callback.
TimePicker *	Use when no TimePicker or PopupTimePicker is on the initial page but will be added during the callback.
SpecialDates	Use when no SpecialDates control is on the initial page but will be added during the callback.
ContextMenu	Use when no ContextMenu control is on the initial page but will be added during the

	callback.
FieldStateControllers	Use when no type of FieldStateController is on the initial page but will be added during a callback. This includes FieldStateController, MultiFieldStateController, FSCOnCommand, and MultiFSCOnCommand.
CalculationControllers	Use when no CalculationController is on the initial page but will be added during a callback.
TextCounter	Use when no TextCounter is on the initial page but will be added during the callback.
AutoPostBack	Use when you validate on AutoPostBack.
ChangeMonitor	Use when the ChangeMonitor is added to the page after postback.
SubmitControls	Use when no Button, LinkButton, or ImageButton is on the initial page but will be added during the callback.
DisableOnSubmit	Use when no Button uses the DisableOnSubmit feature on the initial page but will be used during a callback.
Popups	Use when you create a control using the Popup code.

* The next step is required for these controls.

- For any of the following items from the above chart: DateTextBoxes, TimeTextBoxes, Calendar, MultiSelectionCalendar, MonthYearPicker, and TimePicker, their popup features must be preloaded during the initial page load, even though the control itself is not shown until the AJAX update.

Follow these instructions for *each type of control* that was identified:

- Create a new instance of the control to the page. Put it just after the <form> tag in the web form (aspx) file or at the beginning of the UserControl. This will avoid having it enclosed in another web control that may have its **Visible** property set to false.

```
<form id="form1" runat="server" >
  <des:DateTextBox id="DummyDTB" runat="server" />
  <asp:othercontrols here />
```

Make sure this control is always added to the page, even if you are not sure if same type of control will actually be added during a callback. **WARNING:** It must be added to the Page's Control tree before any of the same type controls that you plan to show in the AJAX panel because its PreRender method must run first. Otherwise, you will get JavaScript errors after the callback.

- Set up the popup elements on this control the way you want to see them when the control is shown in the browser. These popups will be written out when the page is first generated.

```
<des:DateTextBox id="DummyDTB" runat="server" >
  <PopupCalendar>
    <Calendar propertyname="value" propertyname2="value" />
  </PopupCalendar>
</des:DateTextBox>
```

- Call the new control's PreLoadForAJAX() method in Page_Load().

```
DummyDTB.PreLoadForAJAX( )
```

The call to PreLoadForAJAX() prevents this control from appearing on the page, but DES will use it to preload the HTML and scripts of its popups.

See "Example 4 – Control with Popups first created during Callback".

Example 2 – Identify individual DES Controls for Optimized Performance

Suppose you have a WebAsyncRefreshPanel control that updates a DES TextBox and an associated RequiredTextValidator when a DES Button is clicked. You want to use the **InAJAXUpdate** property on each DES control in the UpdatePanel because there are other DES controls on the page.

*Note: To analyze how the **InAJAXUpdate** property on DES's controls have been set up, see "Analyzing the InAJAXUpdate Properties on DES Controls".*

```
<igmisc:WebAsyncRefreshPanel id="WebAsyncRefreshPanel1" runat="server">
<des:TextBox id="TextBox1" runat="server" InAJAXUpdate="true" />&nbsp;
<des:RequiredTextValidator id="RTV1" runat="server" InAJAXUpdate="true"
ControlIDToEvaluate="TextBox1" ErrorMessage="Required" /><br />
<des:Button id="Submit" runat="server" Text="Submit" InAJAXUpdate="true" />
</igmisc:WebAsyncRefreshPanel>
```

When using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
AJAXFramework="InfragisticsAJAX" AJAXControlID="WebAsyncRefreshPanel1" >
</des:PageManager>
```

Programmatically:

This code goes in Page_Load():

[C#]

```
PeterBlum.DES.AJAXManager.UsingInfragisticsAJAX(WebAsyncRefreshPanel1);
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingInfragisticsAJAX(WebAsyncRefreshPanel1)
```


Example 4 – Control with Popups first created during Callback

Suppose you have a WebAsyncRefreshPanel control with GridView that shows a DateTextBox in edit mode. Until the GridView is in edit mode, the <EditItemTemplate> is not created, requiring a dummy DateTextBox to preload the popups. The dummy DateTextBox's popups must have the same properties as the actual DateTextBox. In this case, **CssClass** is set to a custom style sheet class.

```
<des:DateTextBox id="DummyDTB" runat="server" >
  <PopupCalendar>
    <Calendar CssClass="MyCalendarClass" />
  </PopupCalendar>
</des:DateTextBox>

<igmisc:WebAsyncRefreshPanel id="WebAsyncRefreshPanel1" runat="server">
  <asp:GridView id="GridView1" runat="server" OnRowCreated="GridView1_RowCreated" >
    <Columns>
      <des:CommandField ShowEditButton="True" />
      <asp:TemplateField>
        <ItemTemplate>...</ItemTemplate>
        <EditItemTemplate>
          <des:DateTextBox id="DateTextBox1" runat="server" InAJAXUpdate="true" >
            <PopupCalendar>
              <Calendar CssClass="MyCalendarClass" />
            </PopupCalendar>
          </des:DateTextBox>
        </EditItemTemplate>
      </asp:TemplateField>
    </Columns>
  </asp:GridView>
</igmisc:WebAsyncRefreshPanel>
```

Using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
  AJAXFramework="InfragisticsAJAX" AJAXControlID="WebAsyncRefreshPanel1"
  PreLoadForAJAX-DateTextBoxes="True" >
</des:PageManager>
```

This code goes in Page_Load ():

[C#]

```
DummyDTB.PreLoadForAJAX ( ) ;
```

[VB]

```
DummyDTB.PreLoadForAJAX ( )
```

Programmatically:

This code goes in Page_Load ():

[C#]

```
PeterBlum.DES.AJAXManager.UsingInfragisticsAJAX(WebAsyncRefreshPanel1);
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
  PeterBlum.DES.AJAXManager.FeatureList.DateTextBoxes);
DummyDTB.PreLoadForAJAX ( ) ;
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingInfragisticsAJAX(WebAsyncRefreshPanel1)
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _
    PeterBlum.DES.AJAXManager.FeatureList.DateTextBoxes)
DummyDTB.PreLoadForAJAX()
```

Using ComponentArt Callback

<http://www.componentart.com/>

ALERT: Peter's Data Entry Suite was tested with ComponentArt 2006 Q1.

ComponentArt Callback is a control that applies AJAX to controls you provide it within its **Callback** event handler method. DES controls are setup in the Callback event handler like others, by calling their `RenderControl()` method. There is additional setup required. Use the steps here to correctly setup DES to work with ComponentArt Callback.

ALERT: You are virtually guaranteed **JavaScript errors** and **lack of functionality** if you do not implement these steps correctly.

1. For any page with a ComponentArt Callback control that contains or is in some way connected to a DES control, identify it.

When using the PageManager:

Set **AJAXFramework** to `ComponentArtCallback` and **AJAXControlID** to the ID of the Callback control. *Note that these controls must be in the same naming container. Note: This only supports a single ComponentArt Callback control. If you have more, use the programmatic method below.*

```
<des:PageManager runat="server" AJAXFramework="ComponentArtCallback"
AJAXControlID="CallbackControlID" />
```

Programmatically:

Call this method in `Page_Load()` for each Callback control:

```
PeterBlum.DES.AJAXManager.UsingComponentArtCallback(CallbackControl)
```

ALERT: For any code that uses the `PeterBlum.DES.AJAXManager`, it must run every time `Page_Load()` is called. Do not nest it inside of an IF statement like `If IsPostBack = False Then`.

2. Identify which DES controls are participating in the AJAX update. Aside from those inside of an AJAX-enabled control, include those that point to controls involved in an AJAX update. Examples: Validators pointing to data entry controls, Enabler conditions, FieldStateControllers, CalculationControllers, and NativeControlExtenders.

There are two approaches:

- Quick Setup – Use a single setting to establish that every DES control on the page as involved in the AJAX update (even if its not). It provides a fast setup but will make AJAX updates perform less than optimally because all DES controls will transmit their data.
- Optimized Performance – Identify individual controls involved in the AJAX update. Allows only those controls to transmit their HTML and JavaScript. This can greatly improve performance during a callback.

Recommendation: Use Quick Setup during initial page design. If most or all DES controls are involved in an AJAX update, retain this set up. Otherwise, switch to the Optimized performance setup.

They are both discussed below.

Quick Setup

When using the PageManager control:

Set the **AllInAJAXUpdate** property to `true`.

```
<des:PageManager runat="server"
AJAXFramework="ComponentArtCallback" AllInAJAXUpdate="True" />
```

See “Example 1 – Using the AllInAJAXUpdate property for “Quick Setup””

Programmatically:

Set the **PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate** property to `true` in `Page_Load()`. This will tell all DES controls to transmit their data during each AJAX callback.

```
PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate = true
```

Optimized performance

Every DES control has the **InAJAXUpdate** property. Set it to `true`. For example:

```

<des:RequiredTextValidator InAJAXUpdate="true" [other properties] />
<des:TextBox InAJAXUpdate="true" [other properties] />
<des:FieldStateController InAJAXUpdate="true" [other properties] />
<des:Button InAJAXUpdate="true" [other properties] />
<des:CalculationController InAJAXUpdate="true" [other properties] />

```

*Note: To analyze how the **InAJAXUpdate** property on DES's controls have been set up, see "Analyzing the InAJAXUpdate Properties on DES Controls".*

See "Example 2 – Identify individual DES Controls for Optimized Performance".

Visual Studio and Visual Web Developer design mode suggestions:

- Select all DES controls that are in the AJAX update. The Properties Editor will display all properties they have in common, including **InAJAXUpdate**. Set it to `true` to update all at once.
- *ASP.NET 2 only.* Each DES control has a SmartTag with the setting **Involved in an AJAX Update**. Mark its checkbox.

To let the computer do some of the work, use the

`PeterBlum.DES.AJAXManager.SetChildrenInAJAXUpdate()` method to update a container control, like a panel, so that all of its child DES controls have their **InAJAXUpdate** property set to `true`. If any DES control points to another control outside of the container, its **InAJAXUpdate** property will also be set to `true`. However, if there is a DES control outside the container that is not pointed to by a control inside the container, it will still need you to set **InAJAXUpdate**. See "PeterBlum.DES.AJAXManager.SetChildrenInAJAXUpdate".

3. You should already have a **Callback** event handler defined on your page. It should call the `RenderContent()` method of each control it will update with AJAX. In addition, add a call to `PeterBlum.DES.AJAXManager.OutputToComponentArtCallback()` at the end, as shown here:

[C#]

```

private void Callback1_Callback(object sender,
    ComponentArt.Web.UI.CallbackEventArgs e)
{
    DateTextBox1.RenderControl(e.Output);
    More controls calling RenderControl
    PeterBlum.DES.AJAXManager.OutputToComponentArtCallback(e.Output);
}

```

[VB]

```

Private Sub Callback1_Callback(ByVal sender As Object, _
    ByVal e As ComponentArt.Web.UI.CallbackEventArgs) _
    Handles Callback1.Callback
    DateTextBox1.RenderControl(e.Output)
    More controls calling RenderControl
    PeterBlum.DES.AJAXManager.OutputToComponentArtCallback(e.Output)
End Sub

```

4. If a type of DES control or feature is not initially added to page but will later be added during a callback, it needs to be identified when the page is initially requested (`Page.IsPostBack = false`). Examples include a `DataGrid` or `GridView` that switches to edit mode with textboxes and validators; and the `Wizard` which often does not have data entry controls on the first step.

When using the `PageManager` control:

Using the chart below, set the desired property to true on the `PageManager.PreLoadForAJAX` property. For example:

```
<des:PageManager runat="server" PreLoadForAJAX-Validators="True"
PreLoadForAJAX-TextBoxes="True" />
```

Programmatically:

Call `PeterBlum.DES.AJAXManager.PreregisterForAJAX()`.

You pass the enumerated type `PeterBlum.DES.AJAXManager.FeatureList`, whose values are listed in the chart below. For example:

```
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.Validators)
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.TextBoxes)
```

Note: It is safe to call `PreregisterForAJAX()` even when the controls are shown when the page is first generated. All it does is guarantee scripts are added to the page. If you call it when the controls will never be shown, it adds the overhead of loading script files and setting up small global objects.

The `PreregisterForAJAX()` method also accepts an array of `FeatureList` values like this:

[C#]

```
using PeterBlum.DES;
...
AJAXManager.FeatureList[] vFeatures =
    {AJAXManager.FeatureList.Validators, AJAXManager.FeatureList.TextBoxes};
AJAXManager.PreregisterForAJAX(vFeatures);
```

[VB]

```
Imports PeterBlum.DES
...
Dim vFeatures() As AJAXManager.FeatureList = New AJAXManager.FeatureList() _
    {AJAXManager.FeatureList.Validators, AJAXManager.FeatureList.TextBoxes}
AJAXManager.PreregisterForAJAX(vFeatures)
```

See “Example 3 – Controls added during Callback”.

THE CHART OF FEATURES IS ON THE NEXT PAGE

AJAXManager.FeatureList values and PageManager.PreLoadForAJAX properties	
Validators	Use when no validator is on the initial page but will be added during a callback.
ValidationSummary	Use when no ValidationSummary is on the initial page but will be added during a callback.
CombinedErrorMessages	Use when a CombinedErrorMessages control is added on a callback.
TextBoxes	Use when no DES TextBox is on the initial page but will be added during a callback. This includes TextBox, FilteredTextBox, IntegerTextBox, DecimalTextBox, PercentTextBox, and CurrencyTextBox. Also consider Hints, AutoPostBack.
MultiSegmentDataEntry	Use when no MultiSegmentDataEntry control is on the initial page but will be added during a callback. Also consider Hints.
Hints (Also when using ToolTips)	Use when no hints are on the initial page but will be used during a callback. This also applies when using the hint properties on DES's textboxes and MultiSegmentDataEntry controls. By default, ToolTips are converted to hints. When using ToolTips, you need to preload the Hints feature or to turn off the conversion of ToolTips to hints with the ToolTipsAsHints property on the PageManager control or PeterBlum.DES.Globals.Page . See "Properties on the PeterBlum.DES.Globals.Page.HintManager Property" topic in the Interactive Pages Guide .
DateTextBoxes *	Use when no DateTextBox, AnniversaryTextBox, or MonthYearTextBox is on the initial page but will be added during the callback. Also consider Hints, AutoPostBack.
TimeTextBoxes *	Use when no TimeOfDayTextBox or DurationTextBox is on the initial page but will be added during the callback. Also consider Hints, AutoPostBack.
Calendar *	Use when no Calendar is on the initial page but will be added during the callback.
MultiSelectionCalendar*	Use when no MultiSelectionCalendar is on the initial page but will be added during the callback.
MonthYearPicker *	Use when no MonthYearPicker or PopupMonthYearPicker is on the initial page but will be added during the callback.
TimePicker *	Use when no TimePicker or PopupTimePicker is on the initial page but will be added during the callback.
SpecialDates	Use when no SpecialDates control is on the initial page but will be added during the callback.
ContextMenu	Use when no ContextMenu control is on the initial page but will be added during the callback.
FieldStateControllers	Use when no type of FieldStateController is on the initial page but will be added during a callback. This includes FieldStateController, MultiFieldStateController, FSCOnCommand, and MultiFSCOnCommand.
CalculationControllers	Use when no CalculationController is on the initial page but will be added during a callback.
TextCounter	Use when no TextCounter is on the initial page but will be added during the callback.
AutoPostBack	Use when you validate on AutoPostBack.
ChangeMonitor	Use when the ChangeMonitor is added to the page after postback.
SubmitControls	Use when no Button, LinkButton, or ImageButton is on the initial page but will be added during the callback.
DisableOnSubmit	Use when no Button uses the DisableOnSubmit feature on the initial page but will be used during a callback.
Popups	Use when you create a control using the Popup code.

* The next step is required for these controls.

5. For any of the following items from the above chart: DateTextBoxes, TimeTextBoxes, Calendar, MultiSelectionCalendar, MonthYearPicker, and TimePicker, their popup features must be preloaded during the initial page load, even though the control itself is not shown until the AJAX update.

Follow these instructions for *each type of control* that was identified:

- Create a new instance of the control to the page. Put it just after the <form> tag in the web form (aspx) file or at the beginning of the UserControl. This will avoid having it enclosed in another web control that may have its **Visible** property set to false.

```
<form id="form1" runat="server" >
  <des:DateTextBox id="DummyDTB" runat="server" />
  <asp:othercontrols here />
```

Make sure this control is always added to the page, even if you are not sure if same type of control will actually be added during a callback. **WARNING:** *It must be added to the Page's Control tree before any of the same type controls that you plan to show in the AJAX panel because its PreRender method must run first. Otherwise, you will get JavaScript errors after the callback.*

- Set up the popup elements on this control the way you want to see them when the control is shown in the browser. These popups will be written out when the page is first generated.

```
<des:DateTextBox id="DummyDTB" runat="server" >
  <PopupCalendar>
    <Calendar propertyname="value" propertyname2="value" />
  </PopupCalendar>
</des:DateTextBox>
```

- Call the new control's PreLoadForAJAX() method in Page_Load().

```
DummyDTB.PreLoadForAJAX( )
```

The call to PreLoadForAJAX() prevents this control from appearing on the page, but DES will use it to preload the HTML and scripts of its popups.

See "Example 4 – Control with Popups first created during Callback".

Example 1 – Using the AllInAJAXUpdate property for “Quick Setup”

Suppose you have a ComponentArt Callback control that updates a DES TextBox and an associated RequiredTextValidator when a DES Button is clicked. You choose the **PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate** property because these are the only DES controls on the page.

```
<componentart:Callback id="CallBack1" runat="server">
<content>

<des:TextBox id="TextBox1" runat="server" />&nbsp;
<des:RequiredTextValidator id="RTV1" runat="server"
    ControlIDToEvaluate="TextBox1" ErrorMessage="Required" /><br />
<des:Button id="Submit" runat="server" Text="Submit" />

</content>
</componentart:Callback>
```

When using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
    AJAXFramework="ComponentArtCallback" AJAXControlID="CallBack1"
    AllInAJAXUpdate="True" >
</des:PageManager>
```

Programmatically:

This code goes in Page_Load():

[C#]

```
PeterBlum.DES.AJAXManager.UsingComponentArtCallback(CallBack1);
PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate = true;
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingComponentArtCallback(CallBack1)
PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate = True
```

The Callback Event Handler Method:

[C#]

```
private void Callback1_Callback(object sender,
    ComponentArt.Web.UI.CallbackEventArgs e)
{
    TextBox1.RenderControl(e.Output);
    RTV1.RenderControl(e.Output);
    Button1.RenderControl(e.Output);

    PeterBlum.DES.AJAXManager.OutputToComponentArtCallback(e.Output);
}
```

[VB]

```
Private Sub Callback1_Callback(ByVal sender As Object, _
    ByVal e As ComponentArt.Web.UI.CallbackEventArgs) _
    Handles Callback1.Callback

    TextBox1.RenderControl(e.Output)
    RTV1.RenderControl(e.Output)
    Button1.RenderControl(e.Output)

    PeterBlum.DES.AJAXManager.OutputToComponentArtCallback(e.Output)
End Sub
```

Example 2 – Identify individual DES Controls for Optimized Performance

Suppose you have a ComponentArt Callback control that updates a DES TextBox and an associated RequiredTextValidator when a DES Button is clicked. You want to use the **InAJAXUpdate** property on each DES control in the Callback control because there are other DES controls on the page.

*Note: To analyze how the **InAJAXUpdate** property on DES's controls have been set up, see "Analyzing the InAJAXUpdate Properties on DES Controls".*

```
<componentart:Callback id="Callback1" runat="server">
<content>

<des:TextBox id="TextBox1" runat="server" InAJAXUpdate="true" />&nbsp;
<des:RequiredTextValidator id="RTV1" runat="server" InAJAXUpdate="true"
    ControlIDToEvaluate="TextBox1" ErrorMessage="Required" /><br />
<des:Button id="Submit" runat="server" Text="Submit" InAJAXUpdate="true" />

</content>
</componentart:Callback>
```

When using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
    AJAXFramework="ComponentArtCallback" AJAXControlID="Callback1" >
</des:PageManager>
```

Programmatically:

This code goes in Page_Load():

[C#]

```
PeterBlum.DES.AJAXManager.UsingComponentArtCallback(Callback1);
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingComponentArtCallback(Callback1)
```

The Callback Event Handler Method:

[C#]

```
private void Callback1_Callback(object sender,
    ComponentArt.Web.UI.CallbackEventArgs e)
{
    TextBox1.RenderControl(e.Output);
    RTV1.RenderControl(e.Output);
    Button1.RenderControl(e.Output);

    PeterBlum.DES.AJAXManager.OutputToComponentArtCallback(e.Output);
}
```

[VB]

```
Private Sub Callback1_Callback(ByVal sender As Object, _
    ByVal e As ComponentArt.Web.UI.CallbackEventArgs) _
    Handles Callback1.Callback

    TextBox1.RenderControl(e.Output)
    RTV1.RenderControl(e.Output)
    Button1.RenderControl(e.Output)

    PeterBlum.DES.AJAXManager.OutputToComponentArtCallback(e.Output)
End Sub
```

Example 3 – Controls added during Callback

Suppose you have a ComponentArt Callback control that updates a DES TextBox and an associated RequiredTextValidator when a DES Button is clicked. The TextBox and Validator controls are inside of a Panel which is **Visible=false** until the DES Button sets it to **Visible=true** during a callback.

```
<componentart:CallBack id="CallBack1" runat="server">
<content>

    <asp:Panel id="Panell" runat="server" Visible="false">
        <des:TextBox id="TextBox1" runat="server" InAJAXUpdate="true" />&nbsp;  
        <des:RequiredTextValidator id="RTV1" runat="server" InAJAXUpdate="true"
            ControlIDToEvaluate="TextBox1" ErrorMessage="Required" />
    </asp:Panel>
    <des:Button id="Submit" runat="server" Text="Submit" InAJAXUpdate="true"
        Click="Show_ButtonClick" />

</content>
</componentart:CallBack>
```

When using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
    AJAXFramework="ComponentArtCallback" AJAXControlID="CallBack1"
    PreLoadForAJAX-Validators="True" PreLoadForAJAX-TextBoxes="True" >
</des:PageManager>
```

Programmatically:

This code goes in Page_Load():

[C#]

```
PeterBlum.DES.AJAXManager.UsingComponentArtCallback(CallBack1);
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.Validators);
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.TextBoxes);
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingComponentArtCallback(CallBack1)
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _
    PeterBlum.DES.AJAXManager.FeatureList.Validators)
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _
    PeterBlum.DES.AJAXManager.FeatureList.TextBoxes)
```

This code goes in Show_ButtonClick():

[C#]

```
Panell.Visible = true;
```

[VB]

```
Panell.Visible = True
```

The Callback Event Handler Method:

[C#]

```
private void Callback1_Callback(object sender,
    ComponentArt.Web.UI.CallbackEventArgs e)
{
    Panell1.RenderControl(e.Output); // updates TextBox1 and RTV1
    Button1.RenderControl(e.Output);

    PeterBlum.DES.AJAXManager.OutputToComponentArtCallback(e.Output);
}
```

[VB]

```
Private Sub Callback1_Callback(ByVal sender As Object, _
    ByVal e As ComponentArt.Web.UI.CallbackEventArgs) _
    Handles Callback1.Callback

    Panell1.RenderControl(e.Output) ' updates TextBox1 and RTV1
    Button1.RenderControl(e.Output)

    PeterBlum.DES.AJAXManager.OutputToComponentArtCallback(e.Output)
End Sub
```

Example 4 – Control with Popups first created during Callback

Suppose you have a ComponentArt Callback control with GridView that shows a DateTextBox in edit mode. Until the GridView is in edit mode, the <EditItemTemplate> is not created, requiring a dummy DateTextBox to preload the popups. The dummy DateTextBox's popups must have the same properties as the actual DateTextBox. In this case, **CssClass** is set to a custom style sheet class.

```
<des:DateTextBox id="DummyDTB" runat="server" >
  <PopupCalendar>
    <Calendar CssClass="MyCalendarClass" />
  </PopupCalendar>
</des:DateTextBox>

<componentart:CallBack id="CallBack1" runat="server">
<content>

  <asp:GridView id="GridView1" runat="server" OnRowCreated="GridView1_RowCreated" >
    <Columns>
      <des:CommandField ShowEditButton="True" />
      <asp:TemplateField>
        <ItemTemplate>...</ItemTemplate>
        <EditItemTemplate>

          <des:DateTextBox id="DateTextBox1" runat="server" InAJAXUpdate="true" >
            <PopupCalendar>
              <Calendar CssClass="MyCalendarClass" />
            </PopupCalendar>
          </des:DateTextBox>

        </EditItemTemplate>
      </asp:TemplateField>
    </Columns>
  </asp:GridView>
</content>
</componentart:CallBack>
```

Using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
  AJAXFramework="ComponentArtCallback" AJAXControlID="CallBack1"
  PreLoadForAJAX-DateTextBoxes="True" >
</des:PageManager>
```

This code goes in Page_Load():

[C#]

```
DummyDTB.PreLoadForAJAX();
```

[VB]

```
DummyDTB.PreLoadForAJAX();
```

Programmatically:

This code goes in Page_Load():

[C#]

```
PeterBlum.DES.AJAXManager.UsingComponentArtCallback(CallBack1);
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
  PeterBlum.DES.AJAXManager.FeatureList.DateTextBoxes);
DummyDTB.PreLoadForAJAX();
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingComponentArtCallback(CallBack1)
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _
    PeterBlum.DES.AJAXManager.FeatureList.DateTextBoxes)
DummyDTB.PreLoadForAJAX()
```

The Callback Event Handler Method:

[C#]

```
private void Callback1_Callback(object sender,
    ComponentArt.Web.UI.CallbackEventArgs e)
{
    GridView1.RenderControl(e.Output);

    PeterBlum.DES.AJAXManager.OutputToComponentArtCallback(e.Output);
}
```

[VB]

```
Private Sub Callback1_Callback(ByVal sender As Object, _
    ByVal e As ComponentArt.Web.UI.CallbackEventArgs) _
    Handles Callback1.Callback

    GridView1.RenderControl(e.Output)

    PeterBlum.DES.AJAXManager.OutputToComponentArtCallback(e.Output)
End Sub
```

Using MagicAjax

<http://www.magicajax.net>

ALERT: Peter's Data Entry Suite was built with MagicAjax 0.3.0. Since this is a prerelease product, it's possible that later releases will not work with DES. (It is likely that DES can be fixed to address such changes.)

MagicAjax supplies the AjaxPanel control to replace HTML.

ALERT: You are virtually guaranteed **JavaScript errors and lack of functionality** if you do not implement these steps correctly.

1. For each AjaxPanel that contains any DES controls or controls pointed to by a DES control, identify it.

When using the PageManager:

Set **AJAXFramework** to InfragisticsAJAX and **AJAXControlID** to the ID of the WebAsyncRefreshPanel or UltraWebTab. *Note that these controls must be in the same naming container. Note: This only supports a single AjaxPanel control. If you have more, use the programmatic method below.*

```
<des:PageManager runat="server"
    AJAXFramework="MagicAJAX" AJAXControlID="AjaxPanel" />
```

Programmatically:

Call this method in Page_Load():

```
PeterBlum.DES.AJAXManager.UsingMagicAjax(ajaxpanel)
```

ALERT: For any code that uses the `PeterBlum.DES.AJAXManager`, it must run every time `Page_Load()` is called. Do not nest it inside of an IF statement like `If IsPostBack = False Then`.

2. Identify which DES controls are participating in the AJAX update. Aside from those inside of an AJAXPanel, include those that point to controls inside of AJAXPanels. Examples: Validators pointing to data entry controls, Enabler conditions, FieldStateControllers, CalculationControllers, and NativeControlExtenders.

There are two approaches:

- Quick Setup – Use a single setting to establish that every DES control on the page as involved in the AJAX update (even if its not). It provides a fast setup but will make AJAX updates perform less than optimally because all DES controls will transmit their data.
- Optimized Performance – Identify individual controls involved in the AJAX update. Allows only those controls to transmit their HTML and JavaScript. This can greatly improve performance during a callback.

Recommendation: Use Quick Setup during initial page design. If most or all DES controls are involved in an AJAX update, retain this set up. Otherwise, switch to the Optimized performance setup.

They are both discussed below.

Quick Setup

When using the PageManager control:

Set the **AllInAJAXUpdate** property to true.

```
<des:PageManager runat="server"
    AJAXFramework="MicrosoftAJAX" AllInAJAXUpdate="True" />
```

Programmatically:

Set the **PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate** property to true in `Page_Load()`. This will tell all DES controls to transmit their data during each AJAX callback.

```
PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate = true
```

See “Example 1 – Using the AllInAJAXUpdate property for “Quick Setup””.

Optimized performance

Every DES control has the **InAJAXUpdate** property. Set it to `true`. For example:

```

<des:RequiredTextValidator InAJAXUpdate="true" [other properties] />
<des:TextBox InAJAXUpdate="true" [other properties] />
<des:FieldStateController InAJAXUpdate="true" [other properties] />
<des:Button InAJAXUpdate="true" [other properties] />
<des:CalculationController InAJAXUpdate="true" [other properties] />

```

*Note: To analyze how the **InAJAXUpdate** property on DES's controls have been set up, see "Analyzing the InAJAXUpdate Properties on DES Controls".*

See "Example 2 – Identify individual DES Controls for Optimized Performance".

Visual Studio and Visual Web Developer design mode suggestions:

- Select all DES controls that are in the AJAX update. The Properties Editor will display all properties they have in common, including **InAJAXUpdate**. Set it to `true` to update all at once.
- *ASP.NET 2 only.* Each DES control has a SmartTag with the setting **Involved in an AJAX Update**. Mark its checkbox.

To let the computer do some of the work, use the

`PeterBlum.DES.AJAXManager.SetChildrenInAJAXUpdate()` method to update a container control, like a panel, so that all of its child DES controls have their **InAJAXUpdate** property set to `true`. If any DES control points to another control outside of the container, its **InAJAXUpdate** property will also be set to `true`. However, if there is a DES control outside the container that is not pointed to by a control inside the container, it will still need you to set **InAJAXUpdate**. See "PeterBlum.DES.AJAXManager.SetChildrenInAJAXUpdate".

3. If a type of DES control or feature is not initially added to page but will later be added during a callback, it needs to be identified when the page is initially requested (`Page.IsPostBack = false`). Examples include a DataGrid or GridView that switches to edit mode with textboxes and validators; and the Wizard which often does not have data entry controls on the first step.

When using the PageManager control:

Using the chart below, set the desired property to true on the **PageManager.PreLoadForAJAX** property. For example:

```

<des:PageManager runat="server" PreLoadForAJAX-Validators="True"
PreLoadForAJAX-TextBoxes="True" />

```

Programmatically:

Call `PeterBlum.DES.AJAXManager.PreregisterForAJAX()`.

You pass the enumerated type `PeterBlum.DES.AJAXManager.FeatureList`, whose values are listed in the chart below. For example:

```

PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.Validators)
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.TextBoxes)

```

Note: It is safe to call `PreregisterForAJAX()` even when the controls are shown when the page is first generated. All it does is guarantee scripts are added to the page. If you call it when the controls will never be shown, it adds the overhead of loading script files and setting up small global objects.

The PreregisterForAJAX() method also accepts an array of FeatureList values like this:

[C#]

```
using PeterBlum.DES;
...
AJAXManager.FeatureList[] vFeatures =
    {AJAXManager.FeatureList.Validators, AJAXManager.FeatureList.TextBoxes};
AJAXManager.PreregisterForAJAX(vFeatures);
```

[VB]

```
Imports PeterBlum.DES
...
Dim vFeatures() As AJAXManager.FeatureList = New AJAXManager.FeatureList() _
    {AJAXManager.FeatureList.Validators, AJAXManager.FeatureList.TextBoxes}
AJAXManager.PreregisterForAJAX(vFeatures)
```

See “Example 3 – Controls added during Callback”.

AJAXManager.FeatureList values and PageManager.PreLoadForAJAX properties	
Validators	Use when no validator is on the initial page but will be added during a callback.
ValidationSummary	Use when no ValidationSummary is on the initial page but will be added during a callback.
CombinedErrorMessages	Use when a CombinedErrorMessages control is added on a callback.
TextBoxes	Use when no DES TextBox is on the initial page but will be added during a callback. This includes TextBox, FilteredTextBox, IntegerTextBox, DecimalTextBox, PercentTextBox, and CurrencyTextBox. Also consider Hints, AutoPostBack.
MultiSegmentDataEntry	Use when no MultiSegmentDataEntry control is on the initial page but will be added during a callback. Also consider Hints.
Hints (Also when using ToolTips)	Use when no hints are on the initial page but will be used during a callback. This also applies when using the hint properties on DES’s textboxes and MultiSegmentDataEntry controls. By default, ToolTips are converted to hints. When using ToolTips, you need to preload the Hints feature or to turn off the conversion of ToolTips to hints with the ToolTipsAsHints property on the PageManager control or PeterBlum.DES.Globals.Page . See “Properties on the PeterBlum.DES.Globals.Page.HintManager Property” topic in the Interactive Pages Guide .
DateTextBoxes *	Use when no DateTextBox, AnniversaryTextBox, or MonthYearTextBox is on the initial page but will be added during the callback. Also consider Hints, AutoPostBack.
TimeTextBoxes *	Use when no TimeOfDayTextBox or DurationTextBox is on the initial page but will be added during the callback. Also consider Hints, AutoPostBack.
Calendar *	Use when no Calendar is on the initial page but will be added during the callback.
MultiSelectionCalendar*	Use when no MultiSelectionCalendar is on the initial page but will be added during the callback.
MonthYearPicker *	Use when no MonthYearPicker or PopupMonthYearPicker is on the initial page but will be added during the callback.
TimePicker *	Use when no TimePicker or PopupTimePicker is on the initial page but will be added during the callback.
SpecialDates	Use when no SpecialDates control is on the initial page but will be added during the callback.
ContextMenu	Use when no ContextMenu control is on the initial page but will be added during the

	callback.
FieldStateControllers	Use when no type of FieldStateController is on the initial page but will be added during a callback. This includes FieldStateController, MultiFieldStateController, FSCOnCommand, and MultiFSCOnCommand.
CalculationControllers	Use when no CalculationController is on the initial page but will be added during a callback.
TextCounter	Use when no TextCounter is on the initial page but will be added during the callback.
AutoPostBack	Use when you validate on AutoPostBack.
ChangeMonitor	Use when the ChangeMonitor is added to the page after postback.
SubmitControls	Use when no Button, LinkButton, or ImageButton is on the initial page but will be added during the callback.
DisableOnSubmit	Use when no Button uses the DisableOnSubmit feature on the initial page but will be used during a callback.
Popups	Use when you create a control using the Popup code.

* The next step is required for these controls.

- For any of the following items from the above chart: DateTextBoxes, TimeTextBoxes, Calendar, MultiSelectionCalendar, MonthYearPicker, and TimePicker, their popup features must be preloaded during the initial page load, even though the control itself is not shown until the AJAX update.

Follow these instructions for *each type of control* that was identified:

- Create a new instance of the control to the page. Put it just after the <form> tag in the web form (aspx) file or at the beginning of the UserControl. This will avoid having it enclosed in another web control that may have its **Visible** property set to false.

```
<form id="form1" runat="server" >
  <des:DateTextBox id="DummyDTB" runat="server" />
  <asp:othercontrols here />
```

Make sure this control is always added to the page, even if you are not sure if same type of control will actually be added during a callback. **WARNING:** It must be added to the Page's Control tree before any of the same type controls that you plan to show in the AJAX panel because its PreRender method must run first. Otherwise, you will get JavaScript errors after the callback.

- Set up the popup elements on this control the way you want to see them when the control is shown in the browser. These popups will be written out when the page is first generated.

```
<des:DateTextBox id="DummyDTB" runat="server" >
  <PopupCalendar>
    <Calendar propertyname="value" propertyname2="value" />
  </PopupCalendar>
</des:DateTextBox>
```

- Call the new control's PreLoadForAJAX() method in Page_Load().

```
DummyDTB.PreLoadForAJAX()
```

The call to PreLoadForAJAX() prevents this control from appearing on the page, but DES will use it to preload the HTML and scripts of its popups.

See "Example 4 – Control with Popups first created during Callback".

Example 1 – Using the AllInAJAXUpdate property for “Quick Setup”

Suppose you have an AjaxPanel with a DES TextBox, an associated RequiredTextValidator, and a DES Button. You choose the **PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate** property because these are the only DES controls on the page.

```
<ajax:ajaxpanel id="AjaxPanel1" runat="server">
    <des:TextBox id="TextBox1" runat="server" />&nbsp;
    <des:RequiredTextValidator id="RTV1" runat="server"
        ControlIDToEvaluate="TextBox1" ErrorMessage="Required" /><br />
    <des:Button id="Submit" runat="server" Text="Submit" />
</ajax:ajaxpanel>
```

When using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
    AJAXFramework="MagicAJAX" AJAXControlID="AjaxPanel1"
    AllInAJAXUpdate="True" >
</des:PageManager>
```

Programmatically:

This code goes in Page_Load():

[C#]

```
PeterBlum.DES.AJAXManager.UsingMagicAjax(AjaxPanel1);
PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate = true;
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingMagicAjax(AjaxPanel1)
PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate = True
```

Example 2 – Identify individual DES Controls for Optimized Performance

Suppose you have an AjaxPanel with a DES TextBox, an associated RequiredTextValidator, and a DES Button. You want to use the **InAJAXUpdate** property on each DES control in the AjaxPanel because there are other DES controls on the page.

*Note: To analyze how the **InAJAXUpdate** property on DES's controls have been set up, see "Analyzing the InAJAXUpdate Properties on DES Controls".*

```
<ajax:ajaxpanel id="AjaxPanel1" runat="server">
  <des:TextBox id="TextBox1" runat="server" InAJAXUpdate="true" />&nbsp;
  <des:RequiredTextValidator id="RTV1" runat="server" InAJAXUpdate="true"
    ControlIDToEvaluate="TextBox1" ErrorMessage="Required" /><br />
  <des:Button id="Submit" runat="server" Text="Submit" InAJAXUpdate="true" />
</ajax:ajaxpanel>
```

When using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
  AJAXFramework="MagicAJAX" AJAXControlID="AjaxPanel1" >
</des:PageManager>
```

Programmatically:

This code goes in Page_Load():

[C#]

```
PeterBlum.DES.AJAXManager.UsingMagicAjax(AjaxPanel1);
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingMagicAjax(AjaxPanel1)
```

Example 3 – Controls added during Callback

Suppose you have an AjaxPanel with a DES TextBox, an associated RequiredTextValidator, and a DES Button. The TextBox and Validator controls are inside of a Panel which is **Visible=false** until the DES Button sets it to **Visible=true** during a callback.

```
<ajax:ajaxpanel id="AjaxPanel1" runat="server">
    <asp:Panel id="Panel1" runat="server" Visible="false">
    <des:TextBox id="TextBox1" runat="server" InAJAXUpdate="true" />&nbsp;
    <des:RequiredTextValidator id="RTV1" runat="server" InAJAXUpdate="true"
    ControlIDToEvaluate="TextBox1" ErrorMessage="Required" /><br />
    </asp:Panel>
    <des:Button id="Show" runat="server" Text="Submit" InAJAXUpdate="true"
    Click="Show_ButtonClick" />
</ajax:ajaxpanel>
```

When using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
    AJAXFramework="MagicAJAX" AJAXControlID="AjaxPanel1"
    PreLoadForAJAX-Validators="True" PreLoadForAJAX-TextBoxes="True" >
</des:PageManager>
```

Programmatically:

This code goes in Page_Load():

[C#]

```
PeterBlum.DES.AJAXManager.UsingMagicAjax(AjaxPanel1);
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.Validators);
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.TextBoxes);
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingMagicAjax(AjaxPanel1)
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _
    PeterBlum.DES.AJAXManager.FeatureList.Validators)
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _
    PeterBlum.DES.AJAXManager.FeatureList.TextBoxes)
```

This code goes in Show_ButtonClick():

[C#]

```
Panel1.Visible = true;
```

[VB]

```
Panel1.Visible = True
```

Example 4 – Control with Popups first created during Callback

Suppose you have an AjaxPanel with GridView that shows a DateTextBox in edit mode. Until the GridView is in edit mode, the <EditItemTemplate> is not created, requiring a dummy DateTextBox to preload the popups. The dummy DateTextBox's popups must have the same properties as the actual DateTextBox. In this case, **CssClass** is set to a custom style sheet class. *Reminder: The Dummy control must not be inside the AjaxPanel. Try to get it just below the <form> tag.*

```
<des:DateTextBox id="DummyDTB" runat="server" >
  <PopupCalendar>
    <Calendar CssClass="MyCalendarClass" />
  </PopupCalendar>
</des:DateTextBox>
<ajax:ajaxpanel id="AjaxPanel1" runat="server">

  <asp:GridView id="GridView1" runat="server" OnRowCreated="GridView1_RowCreated" >
    <Columns>
      <des:CommandField ShowEditButton="True" />
      <asp:TemplateField>
        <ItemTemplate>...</ItemTemplate>
        <EditItemTemplate>

          <des:DateTextBox id="DateTextBox1" runat="server" InAJAXUpdate="true" >
            <PopupCalendar>
              <Calendar CssClass="MyCalendarClass" />
            </PopupCalendar>
          </des:DateTextBox>

        </EditItemTemplate>
      </asp:TemplateField>
    </Columns>
  </asp:GridView>
</ajax:ajaxpanel>
```

Using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
  AJAXFramework="MagicAJAX" AJAXControlID="AjaxPanel1"
  PreLoadForAJAX-DateTextBoxes="True" >
</des:PageManager>
```

This code goes in Page_Load():

[C#]

```
DummyDTB.PreLoadForAJAX();
```

[VB]

```
DummyDTB.PreLoadForAJAX()
```

Programmatically:

This code goes in Page_Load():

[C#]

```
PeterBlum.DES.AJAXManager.UsingMagicAjax(AjaxPanel1);
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
  PeterBlum.DES.AJAXManager.FeatureList.DateTextBoxes);
DummyDTB.PreLoadForAJAX();
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingMagicAjax(AjaxPanell)  
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _  
    PeterBlum.DES.AJAXManager.FeatureList.DateTextBoxes )  
DummyDTB.PreLoadForAJAX( )
```

Using other AJAX Products

Peter's Data Entry Suite supplies the method `PeterBlum.DES.AJAXManager.UsingAJAXUpdates()` to handle AJAX frameworks not directly supported in the previous sections.

- For each control that contains any DES controls or controls pointed to by a DES control, call the `PeterBlum.DES.AJAXManager.UsingAJAXUpdates()` method in `Page_Load()`. Here is the definition of that method.

[C#]

```
static void UsingAJAXUpdates(Page page,
    PeterBlum.DES.AJAXManager.AJAXSystemType systemType,
    bool inCallback)
```

[VB]

```
Shared Sub UsingAJAXUpdates(ByVal page As Page, _
    ByVal systemType As PeterBlum.DES.AJAXManager.AJAXSystemType, _
    ByVal inCallback As Boolean)
```

Parameters

page

The Page object that contains these controls.

systemType

Defines how the AJAX system collects and processes JavaScript within a callback. It may require some experimentation to determine which of these work with your AJAX system. If none of them work, you will not be able to use DES controls with in that AJAX system's control.

To assist you, DES can show alerts during callbacks to show you that its functions are being called. Add this script immediately after your <form> tag:

```
<script language="javascript">
var gDESDebugAJAX = true;
</script>
```

Suggestion: Create a new web form designed to test the AJAX system. Add only a DES TextBox, a RequiredTextValidator, the AJAX system control, and a DES Button to invoke the callback.

Once you determine how to use your AJAX system, please contact Peter at contact@peterblum.com to report your findings. This will allow Peter to provide support for that AJAX system.

The enumerated type `PeterBlum.DES.AJAXManager.AJAXSystemType` has these values:

- `RegisterScripts` – The third party AJAX system automatically collects scripts that were added into the `RegisterStartupScript()` and `RegisterClientScriptBlock()` methods found on the Page (ASP.NET 1.x) or `Page.ClientScript` (ASP.NET 2.0) object.

Note: This technique is used in some second generation AJAX systems, including Microsoft AJAX Library, Atlas and RadAJAX.

- `EmbedScriptsUseFinish` – The third party AJAX system can execute scripts that are embedded inside the HTML block that it replaces. It also has a property that accepts JavaScript, which is run when the callback is completed. When `EmbedScriptsUseFinish` is used, call `PeterBlum.DES.AJAXManager.GetFinishScript()` to get JavaScript code. Assign the code to the property that runs JavaScript when the callback is completed. `GetFinishScript()` takes one parameter, a Boolean. When `true`, the call is made after all validators have been created on the page. When `false`, there may be additional validators created. Generally you call `GetFinishScript()` at the end of `Page_Load()` when `IsPostBack=false` and at the end of your post back event handlers when `IsPostBack=true`. This allows the best opportunity for all validators to be established, so that you can pass `true` as the parameter.

Example

Suppose the AJAX framework uses a control called AjaxPanel. The AjaxPanel class has a property called **ScriptsToRun** that accepts JavaScript to be run when the callback is completed. Your page has an instance of AjaxPanel called AjaxPanel1.

The code shown here is called as the last line in Page_Load() after testing **Page.IsPostBack=false**. It should also be used as the last line of the post back event handlers.

[C#]

```
AjaxPanel1.ScriptsToRun = PeterBlum.DES.AJAXManager.GetFinishScript(true);
```

[VB]

```
AjaxPanel1.ScriptsToRun = PeterBlum.DES.AJAXManager.GetFinishScript(True)
```

- **EmbedScripts** - The third party AJAX system can execute scripts that are embedded inside the HTML block that it replaces. It does not have a way to execute your JavaScript when the callback is completed. This AJAX system type has a key limitation: if the AJAX system executes the scripts as it adds the web control's HTML into the page, that script may be run too early. Often these scripts depend not only on the presence of the web control itself, but another control to initialize with it. All of these controls must have their HTML on the page before the script runs. (**EmbedScriptsUseFinish** is preferred, because it allows you to declare that all HTML is now on the page before running the initialization code.)
- **GetScriptsAtEnd** - The third party control has a way to get JavaScript after the PreRender phase of the controls its updating has run. DES will collect all of the scripts into a string during PreRender. After that is done, call the following method to retrieve the JavaScript code as a string that you pass to the AJAX system:

```
PeterBlum.DES.AJAXManager.GetCachedScripts( )
```

The GetCachedScripts() method returns a string containing the JavaScript.

There are a few ways to get the scripts at the after PreRender runs on the DES controls.

- The AJAX system fires an event handler when it's ready to get the scripts. Use that event handler to call GetCachedScripts().
- *ASP.NET 2.0 only*. The AJAX system can wait until the entire page's OnPreRender process is complete before getting the scripts. Use the OnPreRenderComplete event on the Page object to call GetCachedScripts().
- This is not an exhaustive list. There may be other solutions.

inCallback

Each AJAX system should provide a way to determine if the server side code is executing during a callback. Use that value to set this parameter. When **true**, the server side code is in a callback. When **false**, it is not.

ALERT: For any code that uses the *PeterBlum.DES.AJAXManager*, it must run every time *Page_Load()* is called. Do not nest it inside of an IF statement like *If IsPostBack = False Then*.

2. Identify which DES controls are participating in the AJAX update. Aside from those directly updated, include those that point to controls being updated. Examples: Validators pointing to data entry controls, Enabler conditions, FieldStateControllers, CalculationControllers, and NativeControlExtenders.

There are two approaches:

- **Quick Setup** – Use a single setting to establish that every DES control on the page as involved in the AJAX update (even if its not). It provides a fast setup but will make AJAX updates perform less than optimally because all DES controls will transmit their data.
- **Optimized Performance** – Identify individual controls involved in the AJAX update. Allows only those controls to transmit their HTML and JavaScript. This can greatly improve performance during a callback.

Recommendation: Use Quick Setup during initial page design. If most or all DES controls are involved in an AJAX update, retain this set up. Otherwise, switch to the Optimized performance setup.

They are both discussed below.

Quick Setup

Set the **PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate** property to `true` in `Page_Load()`. This will tell all DES controls to transmit their data during each AJAX callback.

```
PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate = true
```

See “Example 1 – Using the AllInAJAXUpdate property for “Quick Setup””.

Optimized performance

Every DES control has the **InAJAXUpdate** property. Set it to `true`. For example:

```
<des:RequiredTextValidator InAJAXUpdate="true" [other properties] />
<des:TextBox InAJAXUpdate="true" [other properties] />
<des:FieldStateController InAJAXUpdate="true" [other properties] />
<des:Button InAJAXUpdate="true" [other properties] />
<des:CalculationController InAJAXUpdate="true" [other properties] />
```

*Note: To analyze how the **InAJAXUpdate** property on DES's controls have been set up, see “Analyzing the InAJAXUpdate Properties on DES Controls”.*

See “Example 2 – Identify individual DES Controls for Optimized Performance”.

Visual Studio and Visual Web Developer design mode suggestions:

- Select all DES controls that are in the AJAX update. The Properties Editor will display all properties they have in common, including **InAJAXUpdate**. Set it to `true` to update all at once.
- *ASP.NET 2 only.* Each DES control has a SmartTag with the setting **Involved in an AJAX Update**. Mark its checkbox.

To let the computer do some of the work, use the

`PeterBlum.DES.AJAXManager.SetChildrenInAJAXUpdate()` method to update a container control, like a panel, so that all of its child DES controls have their **InAJAXUpdate** property set to `true`. If any DES control points to another control outside of the container, its **InAJAXUpdate** property will also be set to `true`. However, if there is a DES control outside the container that is not pointed to by a control inside the container, it will still need you to set **InAJAXUpdate**. See “PeterBlum.DES.AJAXManager.SetChildrenInAJAXUpdate”.

3. If a type of DES control or feature is not initially added to page but will later be added during a callback, it needs to be identified when the page is initially requested (`Page.IsPostBack = false`). Examples include a `DataGrid` or `GridView` that switches to edit mode with textboxes and validators; and the Wizard which often does not have data entry controls on the first step.

Call `PeterBlum.DES.AJAXManager.PreregisterForAJAX()`.

You pass the enumerated type `PeterBlum.DES.AJAXManager.FeatureList`, whose values are listed in the chart below. For example:

```
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.Validators)
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.TextBoxes)
```

Note: It is safe to call `PreregisterForAJAX()` even when the controls are shown when the page is first generated. All it does is guarantee scripts are added to the page. If you call it when the controls will never be shown, it adds the overhead of loading script files and setting up small global objects.

The PreregisterForAJAX() method also accepts an array of FeatureList values like this:

[C#]

```
using PeterBlum.DES;
...
AJAXManager.FeatureList[] vFeatures =
    {AJAXManager.FeatureList.Validators, AJAXManager.FeatureList.TextBoxes};
AJAXManager.PreregisterForAJAX(vFeatures);
```

[VB]

```
Imports PeterBlum.DES
...
Dim vFeatures() As AJAXManager.FeatureList = New AJAXManager.FeatureList() _
    {AJAXManager.FeatureList.Validators, AJAXManager.FeatureList.TextBoxes}
AJAXManager.PreregisterForAJAX(vFeatures)
```

See “Example 3 – Controls added during Callback”.

AJAXManager.FeatureList values and PageManager.PreLoadForAJAX properties	
Validators	Use when no validator is on the initial page but will be added during a callback.
ValidationSummary	Use when no ValidationSummary is on the initial page but will be added during a callback.
CombinedErrorMessages	Use when a CombinedErrorMessages control is added on a callback.
TextBoxes	Use when no DES TextBox is on the initial page but will be added during a callback. This includes TextBox, FilteredTextBox, IntegerTextBox, DecimalTextBox, PercentTextBox, and CurrencyTextBox. Also consider Hints, AutoPostBack.
MultiSegmentDataEntry	Use when no MultiSegmentDataEntry control is on the initial page but will be added during a callback. Also consider Hints.
Hints (Also when using ToolTips)	Use when no hints are on the initial page but will be used during a callback. This also applies when using the hint properties on DES’s textboxes and MultiSegmentDataEntry controls. By default, ToolTips are converted to hints. When using ToolTips, you need to preload the Hints feature or to turn off the conversion of ToolTips to hints with the ToolTipsAsHints property on the PageManager control or PeterBlum.DES.Globals.Page . See “Properties on the PeterBlum.DES.Globals.Page.HintManager Property” topic in the Interactive Pages Guide .
DateTextBoxes *	Use when no DateTextBox, AnniversaryTextBox, or MonthYearTextBox is on the initial page but will be added during the callback. Also consider Hints, AutoPostBack.
TimeTextBoxes *	Use when no TimeOfDayTextBox or DurationTextBox is on the initial page but will be added during the callback. Also consider Hints, AutoPostBack.
alendar *	Use when no Calendar is on the initial page but will be added during the callback.
MultiSelectionCalendar*	Use when no MultiSelectionCalendar is on the initial page but will be added during the callback.
MonthYearPicker *	Use when no MonthYearPicker or PopupMonthYearPicker is on the initial page but will be added during the callback.
TimePicker *	Use when no TimePicker or PopupTimePicker is on the initial page but will be added during the callback.
SpecialDates	Use when no SpecialDates control is on the initial page but will be added during the callback.
ContextMenu	Use when no ContextMenu control is on the initial page but will be added during the

	callback.
FieldStateControllers	Use when no type of FieldStateController is on the initial page but will be added during a callback. This includes FieldStateController, MultiFieldStateController, FSCOnCommand, and MultiFSCOnCommand.
CalculationControllers	Use when no CalculationController is on the initial page but will be added during a callback.
TextCounter	Use when no TextCounter is on the initial page but will be added during the callback.
AutoPostBack	Use when you validate on AutoPostBack.
ChangeMonitor	Use when the ChangeMonitor is added to the page after postback.
SubmitControls	Use when no Button, LinkButton, or ImageButton is on the initial page but will be added during the callback.
DisableOnSubmit	Use when no Button uses the DisableOnSubmit feature on the initial page but will be used during a callback.
Popups	Use when you create a control using the Popup code.

* The next step is required for these controls.

- For any of the following items from the above chart: DateTextBoxes, TimeTextBoxes, Calendar, MultiSelectionCalendar, MonthYearPicker, and TimePicker, their popup features must be preloaded during the initial page load, even though the control itself is not shown until the AJAX update.

Follow these instructions for *each type of control* that was identified:

- Create a new instance of the control to the page. Put it just after the <form> tag in the web form (aspx) file or at the beginning of the UserControl. This will avoid having it enclosed in another web control that may have its **Visible** property set to false.

```
<form id="form1" runat="server" >
  <des:DateTextBox id="DummyDTB" runat="server" />
  <asp:othercontrols here />
```

Make sure this control is always added to the page, even if you are not sure if same type of control will actually be added during a callback. **WARNING:** It must be added to the Page's Control tree before any of the same type controls that you plan to show in the AJAX panel because its PreRender method must run first. Otherwise, you will get JavaScript errors after the callback.

- Set up the popup elements on this control the way you want to see them when the control is shown in the browser. These popups will be written out when the page is first generated.

```
<des:DateTextBox id="DummyDTB" runat="server" >
  <PopupCalendar>
    <Calendar propertyname="value" propertyname2="value" />
  </PopupCalendar>
</des:DateTextBox>
```

- Call the new control's PreLoadForAJAX() method in Page_Load().

```
DummyDTB.PreLoadForAJAX( )
```

The call to PreLoadForAJAX() prevents this control from appearing on the page, but DES will use it to preload the HTML and scripts of its popups.

See "Example 4 – Control with Popups first created during Callback".

Example 1 – Using the AllInAJAXUpdate property for “Quick Setup”

Suppose AJAX will update a DES TextBox, an associated RequiredTextValidator, and a DES Button. You determine that this system uses the AJAXSystemType of RegisterScripts. It has a control that manages updates called UpdatePanel with a property called **InCallback** which is true when it is in a callback. You choose the **PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate** property because these are the only DES controls on the page.

```
<ajax:UpdatePanel id="UpdatePanel1" runat="server">
  <des:TextBox id="TextBox1" runat="server" />&nbsp;
  <des:RequiredTextValidator id="RTV1" runat="server"
    ControlIDToEvaluate="TextBox1" ErrorMessage="Required" /><br />
  <des:Button id="Submit" runat="server" Text="Submit" />
</ajax:UpdatePanel >
```

This code goes in Page_Load():

[C#]

```
PeterBlum.DES.AJAXManager.UsingAJAXUpdates(Page,
    PeterBlum.DES.AJAXManager.AJAXSystemType.RegisterScripts,
    UpdatePanel1.InCallback);
PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate = true;
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingMagicAjax(AjaxPanel1)
PeterBlum.DES.AJAXManager.Current.AllInAJAXUpdate = True
```

Example 2 – Identify individual DES Controls for Optimized Performance

Suppose AJAX will update a DES TextBox, an associated RequiredTextValidator, and a DES Button. You determine that this system uses the AJAXSystemType of EmbedScripts. It has a control that manages updates called UpdatePanel with a property called **InCallback** which is true when it is in a callback. You want to use the **InAJAXUpdate** property on each DES control in the AjaxPanel because there are other DES controls on the page.

*Note: To analyze how the **InAJAXUpdate** property on DES's controls have been set up, see "Analyzing the InAJAXUpdate Properties on DES Controls".*

```
<ajax:UpdatePanel id="UpdatePanel1" runat="server">
  <des:TextBox id="TextBox1" runat="server" InAJAXUpdate="true" />&nbsp;
  <des:RequiredTextValidator id="RTV1" runat="server" InAJAXUpdate="true"
    ControlIDToEvaluate="TextBox1" ErrorMessage="Required" /><br />
  <des:Button id="Submit" runat="server" Text="Submit" InAJAXUpdate="true" />
</ajax:UpdatePanel>
```

This code goes in Page_Load ():

[C#]

```
PeterBlum.DES.AJAXManager.UsingAJAXUpdates(Page,
    PeterBlum.DES.AJAXManager.AJAXSystemType.EmbedScripts,
    UpdatePanel1.InCallback);
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingAJAXUpdates(Page, _
    PeterBlum.DES.AJAXManager.AJAXSystemType.EmbedScripts, _
    UpdatePanel1.InCallback);
```

Example 3 – Controls added during Callback

Suppose AJAX will update a DES TextBox, an associated RequiredTextValidator, and a DES Button. The TextBox and Validator controls are inside of a Panel which is **Visible=false** until the DES Button sets it to **Visible=true** during a callback. You determine that this system uses the AJAXSystemType of GetScriptsAtEnd. It has a control that manages updates called UpdatePanel with a property called **InCallback** which is true when it is in a callback.

```
<ajax:UpdatePanel id="UpdatePanell" runat="server">
  <asp:Panel id="Panell" runat="server" Visible="false">
    <des:TextBox id="TextBox1" runat="server" InAJAXUpdate="true" />&nbsp;
    <des:RequiredTextValidator id="RTV1" runat="server" InAJAXUpdate="true"
      ControlIDToEvaluate="TextBox1" ErrorMessage="Required" /><br />
  </asp:Panel>
  <des:Button id="Show" runat="server" Text="Submit" InAJAXUpdate="true"
    Click="Show_ButtonClick" />
</ajax:UpdatePanel>
```

This code goes in Page_Load():

[C#]

```
PeterBlum.DES.AJAXManager.UsingAJAXUpdates(Page,
  PeterBlum.DES.AJAXManager.AJAXSystemType.GetScriptsAtEnd,
  UpdatePanell.InCallback);
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
  PeterBlum.DES.AJAXManager.FeatureList.Validators);
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
  PeterBlum.DES.AJAXManager.FeatureList.TextBoxes);
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingAJAXUpdates(Page, _
  PeterBlum.DES.AJAXManager.AJAXSystemType.GetScriptsAtEnd, _
  UpdatePanell.InCallback)
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _
  PeterBlum.DES.AJAXManager.FeatureList.Validators)
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _
  PeterBlum.DES.AJAXManager.FeatureList.TextBoxes)
```

This code goes in Show_ButtonClick():

[C#]

```
Panell.Visible = true;
```

[VB]

```
Panell.Visible = True
```

This code goes after PreRender runs, such as in the Page's PreRenderCompleted event (only in ASP.NET 2.0). In this example, assume that the UpdatePanel class has a property called Scripts where a string of JavaScript can be assigned.

[C#]

```
UpdatePanell.Scripts = PeterBlum.DES.AJAXManager.GetCachedScripts();
```

[VB]

```
UpdatePanell.Scripts = PeterBlum.DES.AJAXManager.GetCachedScripts()
```

Example 4 – Control with Popups first created during Callback

Suppose AJAX will update a GridView that shows a DateTextBox in edit mode. Until the GridView is in edit mode, the <EditItemTemplate> is not created, requiring a dummy DateTextBox to preload the popups. The dummy DateTextBox's popups must have the same properties as the actual DateTextBox. In this case, **CssClass** is set to a custom style sheet class.

You determine that this system uses the AJAXSystemType of EmbedScripts. It has a control that manages updates called UpdatePanel with a property called **InCallback** which is true when it is in a callback. You want to use the **InAJAXUpdate** property on each DES control in the AjaxPanel because there are other DES controls on the page.

```
<des:DateTextBox id="DummyDTB" runat="server" >
  <PopupCalendar>
    <Calendar CssClass="MyCalendarClass" />
  </PopupCalendar>
</des:DateTextBox>
<ajax:UpdatePanel id="UpdatePanel1" runat="server">

  <asp:GridView id="GridView1" runat="server" OnRowCreated="GridView1_RowCreated" >
    <Columns>
      <des:CommandField ShowEditButton="True" />
      <asp:TemplateField>
        <ItemTemplate>...</ItemTemplate>
        <EditItemTemplate>

          <des:DateTextBox id="DateTextBox1" runat="server" InAJAXUpdate="true" >
            <PopupCalendar>
              <Calendar CssClass="MyCalendarClass" />
            </PopupCalendar>
          </des:DateTextBox>

        </EditItemTemplate>
      </asp:TemplateField>
    </Columns>
  </asp:GridView>
</ajax:UpdatePanel>
```

Using the PageManager:

```
<des:PageManager id="PageManager1" runat="server"
  PreLoadForAJAX-DateTextBoxes="True" >
</des:PageManager>
```

This code goes in Page_Load():

[C#]

```
DummyDTB.PreLoadForAJAX();
```

[VB]

```
DummyDTB.PreLoadForAJAX()
```

Programmatically:

This code goes in Page_Load ():

[C#]

```
PeterBlum.DES.AJAXManager.UsingAJAXUpdates( Page,
    PeterBlum.DES.AJAXManager.AJAXSystemType.EmbedScripts,
    UpdatePanell.InCallback );
PeterBlum.DES.AJAXManager.PreregisterForAJAX(
    PeterBlum.DES.AJAXManager.FeatureList.DateTextBoxes );
DummyDTB.PreLoadForAJAX( ) ;
```

[VB]

```
PeterBlum.DES.AJAXManager.UsingAJAXUpdates( Page, _
    PeterBlum.DES.AJAXManager.AJAXSystemType.EmbedScripts, _
    UpdatePanell.InCallback )
PeterBlum.DES.AJAXManager.PreregisterForAJAX( _
    PeterBlum.DES.AJAXManager.FeatureList.DateTextBoxes )
DummyDTB.PreLoadForAJAX( )
```

AJAXManager Properties

The AJAXManager class has several properties that can assist you. Some are static (Shared in Visual Basic), referenced through **PeterBlum.DES.AJAXManager**. Others are found on **PeterBlum.DES.AJAXManager.Current**.

Static (Shared) Properties

- **Current** (PeterBlum.DES.AJAXManager) – A reference to the active AJAXManager class that the page is using. It contains properties you can set to customize the behavior on a page level.

Page-Level Properties

- **AllInAJAXUpdate** (Boolean) – Allow all controls on the page to generate their AJAX-specific scripts. When `true`, it means all controls are in AJAX. When `false`, use the individual control's **InAJAXUpdate**.

It is not optimal to set this to `true` because it may cause more data to be transmitted. However, it's a convenient way to create a page. Later, the user can shut it off and set individual controls to optimize. Or they can use it permanently and take any performance hit.

ALERT: When DES controls are in an AJAX update, you must set AllInAJAXUpdate to true or the individual InAJAXUpdate properties to true on each DES control. Otherwise JavaScript errors will occur after a callback.

- **PageUsesCallbacks** (Boolean, Read Only) – Returns `true` when the page has been set up for callbacks through one of the `AJAXManager.UsingXYZ()` methods.
- **InCallback** (Boolean, Read Only) – Returns `true` if the current page request is in a callback. It is only set after calling one of the `AJAXManager.UsingXYZ()` methods.
- **ReattachAllControls** (Boolean) – Normally the controls inside the panel will have their change monitoring events, `onclick` or `onchange`, hooked up so that changes will invoke DES actions. Controls outside the panel will not.

If the controls inside the panel are used by DES controls outside the panel, set this to `true` to force DES to set up the change monitoring events. It defaults to `false`.

- **RestoreValidatorState** (Boolean) – Callbacks can replace the HTML of validators, eliminating their known state (valid or invalid). Often a callback should have no visual impact on a validator's state. Use **RestoreValidatorState** to preserve or forget the state after a callback. When `true`, it restores the validator state. When `false`, validators appear according to their server side setup which is hidden unless you called `PeterBlum.DES.Globals.Page.Validate()` explicitly. It defaults to `true`.

If your callback does server side processing that changes the fields, such as clearing them to start a new record, you probably should set this to `false`.

When restoring validators, it only affects validators that were previously fired on the form. This prevents validator errors from appearing where they were not present before.

- **EmbedFinishTimeDelay** (Integer) - When using `EmbedScripts` system type through `AJAXManager.UsingAJAXUpdates()` or `UsingRadCallback()`, it must run the function `DES_FinishCallback()` by using a timer. This is the time delay in milliseconds. It defaults to 500 (0.5 seconds). It can be adjusted based on the time it takes the page to normally process the callback.
- **DebugMode** (Boolean) – When `true`, the page will generate several extra scripts useful in debugging: the actual script files and style sheets. Another debugging technique is to set `<% @Page Trace=True %>`. The trace will identify which controls will have their HTML and scripts updated by an AJAX update.

Other AJAXManager Methods

The PeterBlum.DES.AJAXManager class offers these static/shared methods to assist you.

PeterBlum.DES.AJAXManager.SetChildrenInAJAXUpdate

Use the `SetChildrenInAJAXUpdate()` method to update a container control, like a panel, so that all of its child DES controls have their **InAJAXUpdate** property set to `true`. If any DES control points to another control outside of the container, its **InAJAXUpdate** property will also be set to `true`. However, if there is a DES control outside the container that is not pointed to by a control inside the container, it will still need you to set **InAJAXUpdate**.

Call `SetChildrenInAJAXUpdate()` after all child controls are defined. If you define the controls in ASP.NET declarative syntax, you can do this in `Page_Load()`. If controls are created by DataBinding or in a post back event handler, be sure to call this after those processes create their controls.

Be aware that this method must search through the container control's tree of child controls. This can take some time (in terms of server CPU usage) and will happen on *every* page request. It is smart to limit the size of the tree it searches by choosing the smallest container possible and using the *maxDepth* parameter to limit the depth of the tree searched.

You can further optimize by using these steps:

1. Set the Page's **Trace** property to `true`. `<%@ Page Trace="True" %>`
2. Call `SetChildrenInAJAXUpdate()` where appropriate.
3. Open the page in the browser and look at the Trace. It will list each DES control that it had to modify.
4. Set the **InAJAXUpdate** property to `true` on each of those controls.
5. Remove the call to `SetChildrenInAJAXUpdate()` and `Trace="True"`.

Here is the definition.

[C#]

```
void SetChildrenInAJAXUpdate(Control container, int maxDepth);
```

[VB]

```
Sub SetChildrenInAJAXUpdate(ByVal container As Control, _
    ByVal maxDepth As Integer)
```

Parameters

container

The container control. It's often the AJAX framework's "panel" although it can be any container control, such as UserControl, DataGrid, Table, and Panel. (It does not support the radAJAXManager, because its list of controls is not child controls.)

maxDepth

Limits the search down the control tree to optimize the search. Always pass 1 or higher. If you want to search the entire tree, pass 100.

Example

Update controls in the Atlas UpdatePanel control named UpdatePanel1. Limit it to 3 levels of the child controls in the tree.

[C#]

```
PeterBlum.DES.AJAXManager.SetChildrenInAJAXUpdate(UpdatePanel1, 3);
```

[VB]

```
PeterBlum.DES.AJAXManager.SetChildrenInAJAXUpdate(UpdatePanel1, 3)
```

Analyzing the InAJAXUpdate Properties on DES Controls

If you are trying to improve performance or are having controls fail when using the “Good Performance” setup for MS AJAXUpdate, DES provides a tool to expose how *every* DES control’s InAJAXUpdate property is setup.

There are two ways to activate this feature: using a querystring parameter or adding a control to the page.

Querystring parameter

This technique only works when the page is requested the first time. It cannot evaluate any changes made to the page after a callback.

To use it, add the `desdebug=` parameter to the querystring of your URL. You will get a menu. Select the AJAX option.

Note: the `desdebug=` parameter has security restrictions based on IP address. If you are not running from `http://localhost`, you will either have to expand the IP addresses allowed, or define a password. See “Running a DES Debugging Report from when the Server is not local”.

Add a control to the page

This technique takes a little more work but gives you the analysis on every page request, including after the callback.

- Add a Literal or Label control to the page. It should be part of your AJAX update, such as in Microsoft’s UpdatePanel or the RadAJAXManager. *Don’t worry, its temporary.* It will be inserting an extensive element to your page so be prepared for the page to look dramatically different.
- In `Page_Load()`, call this code passing the Literal or Label control:

[C#]

```
PeterBlum.DES.AJAXManager.Current.OutputDescription(control);
```

[VB]

```
PeterBlum.DES.AJAXManager.Current.OutputDescription(control)
```

- Use the page, including callbacks.
- When done, remove the control and line of code, or at least set the control to **Visible=false**.

Using the Analysis

So long as the AllInAJAXUpdate property is false, you will get a report like this:

UniqueID	Type	InAJAXUpdate	Modified*	Notes
TextBox1	IntegerTextBox	False		
Button1	Button	False		
RequiredTextValidator1	RequiredTextValidator	True	In UpdatePanel	
Button2	Button	True	In UpdatePanel	

The InAJAXUpdate column is key here. It should read True for any control that is involved in the AJAX callback or connected to a control in the AJAX callback. When the Modified column has some text, the **InAJAXUpdate** property is initially false but programmatically changed to True. If you are trying to get the best performance, change the **InAJAXUpdate** property to `true` directly on that control.

The String Lookup System

Most string properties throughout DES have an associated LookupID property. When the LookupID property is assigned, you are directing DES to lookup the string within the String Lookup System.

The String Lookup System can be used in two ways:

- Localize strings based on the current culture defined in the **PeterBlum.DES.Globals.Page.CultureInfo** property. DES does not handle translating text. It only provides a way to lookup strings from a data source. There are numerous resources available to translate the text and put it into your data source. There are software products and companies that specialize in this work. A Google search on “language translation software” provides numerous helpful entries.

ASP.NET 2 also provides a mechanism for string lookups using resources called “Declarative Localization Expressions”. It is best suited for ASP.NET Declarative Syntax while the DES String Lookup System also works programmatically.
- Provide a library of standardized strings for validator error messages and anything else you want. The library keeps all the strings in one place where programmers who add Validators to the page only need to identify which string to use. This way, you can edit the strings on one place and change your application globally.

Click on any of these topics to jump to them:

- ◆ Datasources
- ◆ Elements Needed In Your DataSource
- ◆ Using the Resource Manager
 - Mapping the String Group names to the associated resource files.
 - Add .resx Files for each String Group to your Web Application
 - Supporting Cultures
 - Entering Strings and Compiling Them
 - Using non-default Resource
- ◆ Using a Database
 - Set up a Database With The Appropriate Tables
 - Add the DESLookupString Stored Procedure
 - Provide the ConnectionString to the Database
- ◆ Writing Your Own Lookup String Event Handler
 - Create the Event Handler Method
 - Using the OnLookupString property
- ◆ Calling The String Lookup System

Datasources

You must supply a datasource for your strings. Microsoft created the Resource Manager with its .resx files as a convenient way of defining strings. You can define tables in a database, use a third party localization system, or build something and call it through an event handler.

- Resource Manager – Microsoft designed the Resource Manager to handle localization. You create a .resx file, defining a unique ID in the Name attribute for each string and the default or localized text in the Value attribute. You define separate .resx files for each culture, giving them names that differ by the culture name itself: MyMessages.resx, MyMessages.en-US.resx, MyMessages.en-GB.resx, etc. See [http://msdn2.microsoft.com/en-us/library/aa309421\(VS.71\).aspx](http://msdn2.microsoft.com/en-us/library/aa309421(VS.71).aspx) for details. DES provides a framework to quickly start using the Resource Manager as your datasource.

See “Using the Resource Manager”.

- Database – You can use a database to store each string. You define the tables and write a stored proc that DES will call to lookup a string.

See “Using a Database”.

- Third Party Localization Systems – There are several .Net software products that handle localization. Underlying them is a datasource. You can use their tools to store the localized strings. You write an event handler to access their datasource.

See “Writing Your Own Lookup String Event Handler”

- Other solutions – You can invent your own system. Perhaps you have an XML file format or COM object which maintains the localized strings. You write an event handler to access their datasource.

See “Writing Your Own Lookup String Event Handler”

Elements Needed In Your DataSource

When DES needs to lookup a string, it supplies three data elements to a method that will find a match and return the string. Your datasource should be able to handle these elements.

- **LookupID** – This is an identifier used to find a particular string. It is the same value you assign to any DES property that ends in “LookupID”. When using the Resource Manager, this maps to the Name attribute. LookupID values should be unique across a culture.
- **Culture Name** – This is a unique name defining the region and language as defined in the RFC 1766 standard and is found in CultureInfo.Name. See [http://msdn2.microsoft.com/en-us/library/87k6sx8t\(vs.71\).aspx](http://msdn2.microsoft.com/en-us/library/87k6sx8t(vs.71).aspx) for details on this naming convention. You define translated strings for each desired culture, identified with the same LookupID. There are three formats for a culture name:
 - **Language and region specific** – These match the CultureInfo.Name property exactly. Use this for strings where regions use the same overall language but have variations in some words. For example, “color” and “colour” are the same word in English (“en”) but there regional differences where “color” is used in the United States (“en-US”) and “colour” is used in Great Britain (“en-GB”).

The culture name has this format: <language>-<region>. Select as many of these as you want to support. See [http://msdn2.microsoft.com/en-us/library/87k6sx8t\(vs.71\).aspx](http://msdn2.microsoft.com/en-us/library/87k6sx8t(vs.71).aspx) for a list of culture names.

 - **Language only** – These match the <language> part of the CultureInfo.Name property. This allows you to translate for each language without taking into account the differences in regions. Use this for text that is common amongst the regions or is a fallback for when the language and region specific resources do not supply the localized string.
 - **Default** – This is the fall back for all string lookups. It has no culture name. This is the only type used when you are not localizing strings. When you are localizing strings, this is the fallback when there are no matches in the previous two types.
- **String Group** – DES lets you separate your strings into groups so that you can keep lists of closely related data together. When using resources, separate files are used for each group.
 - **ErrorMessages** – Validator.ErrorMessage and Validator.SummaryErrorMessage properties. Also includes PopupErrorFormatter.ErrorMessageHelpLookupID.
 - **Labels** - LocalizableLabel.TextLookupID and the Validator.Label.TextLookupID properties.
 - **ValidationMisc** – All properties on ValidationSummary, NoErrorFormatter, RequiredFieldMarker, and RequiredFieldDescriptors. Properties on ErrorFormatters except for tooltips and hints.
 - **DateTime** – All Date and Time module properties except HintLookupID, HintHelpLookupID and TooltipLookupID on the TextBoxes, which are in the Hints Group.
 - **ConfirmMessages** – For confirm message boxes.
 - **PopupViews** – For elements in the PopupView system that are not the actual message. (Messages come from ErrorMessages, Hints, and ToolTips)
 - **Hints** – For HintLookupID, HintHelpLookupID, and TooltipLookupID properties on various textboxes, buttons, and the NativeControlExtender. It also handles hints added to ContextMenus.
 - **ContextMenus** – ContextMenu properties, except the ConfirmMessageLookupID which is in the ConfirmMessages Group.
 - **TextCounter** – TextCounter control properties.
 - **Misc** – For strings not handled by any DES controls. The user can call the String Lookup System to request strings from this file and assign the results to other controls.

Using the Resource Manager

The .Net Resource Manager is designed to manage localized text. This section will explain how to use resource files with DES.

Visual Studio makes using resources easy. It provides the editors to enter your strings and compiles them automatically. The compiled resources are placed into the same assembly as the web application itself. All other development tools, including Web Matrix, require much more work to set up and manage resources. This documentation provides separate instructions for Visual Studio.Net and other development tools.

Here are the steps to using the Resource Manager:

1. Add .resx files for each String Group to your web application. *DES provides you with initial files.*
2. Determine the cultures that you will support. Create .resx files for each culture within that String Group.
3. Enter strings into the .resx files and compile them into an assembly.
4. Configure DES to use your resources.

Click on any of these topics to jump to them:

- ◆ [Mapping the String Group names to the associated resource files](#)
- ◆ [Add .resx Files for each String Group to your Web Application](#)
- ◆ [Supporting Cultures](#)
- ◆ [Entering Strings and Compiling Them](#)
- ◆ [Using non-default Resource](#)

Mapping the String Group names to the associated resource files

DES supplies resource files for its String Groups. You can use them or your own. The next section documents how to install them. This section helps you map the String Group to the associated resource file name.

String Group Name	File Name	Usage
ErrorMessages	DESErrorMessages.resx	Validator.ErrorMessage and Validator.SummaryErrorMessage properties. Also includes PopupErrorFormatter.ErrorMessageHelpLookupID.
Labels	DESLabels.resx	LocalizableLabel.TextLookupID and the Validator.Label.TextLookupID properties.
ValidationMisc	DESValidationMisc.resx	All properties on ValidationSummary, NoErrorFormatter, RequiredFieldMarker, and RequiredFieldDescriptors. Properties on ErrorFormatters except for tooltips and hints.
DateTime	DESDateTime.resx	All Date and Time module properties except HintLookupID, HintHelpLookupID and TooltipLookupID on the TextBoxes, which are in the Hints Group.
ConfirmMessages	DESConfirmMessages.resx	For confirm message boxes.
PopupViews	DESPopupViews.resx	For elements in the PopupView system that are not the actual message. (Messages come from ErrorMessages, Hints, and ToolTips)
Hints	DESHints.resx	For HintLookupID, HintHelpLookupID, and ToolTipLookupID properties on various textboxes, buttons, and the NativeControlExtender. It also handles hints added to ContextMenus.
ContextMenus	DESContextMenu.resx	ContextMenu properties, except the ConfirmMessageLookupID which is in the ConfirmMessages Group.
TextCounter	DESTextCounter.resx	TextCounter – TextCounter control properties
Misc	DESMisc.resx	For strings not handled by any DES controls. The user can call the String Lookup System to request strings from this file and assign the results to other controls

Add .resx Files for each String Group to your Web Application

DES provides empty .resx files for you to add to your web application. They are in the [DES Product folder]\StringLookup. There is one file for each String Group.

Adding These Files In Visual Studio 2005/2008

If you are using Visual Studio 2005 to develop your web application, here are the steps to add each of these files.

Note: If you are using the Web Application Projects feature of Visual Studio, see the next section.

1. Open Solution Explorer.
2. Determine if you already have a folder called **App_GlobalResources**. If not, right click on the web application project name and select **Add Folder; App_GlobalResources**.
3. Right click on the **App_GlobalResources** folder and select **Add; Add Existing Item**.
4. Add all files of the pattern **DES*.resx** from the [DES Product folder]\StringLookup folder. To learn about these files, see “Mapping the String Group names to the associated resource files”.
5. If you want to support cultures, duplicate these files and rename the according to the naming convention for culture oriented resources. See “Supporting Cultures”.

Adding These Files In Visual Studio 2002-2003 and Web Application Projects in VS2005/8

If you are using Visual Studio.Net 2002 or 2003 to develop your web application, here are the steps to add each of these files.

Note: Each of these files will be added to the root folder of your web application.

1. Open Solution Explorer.
2. Right click on the web application project name and select **Add; Add Existing Item**.
3. Add all files of the pattern **DES*.resx** from the [DES Product folder]\StringLookup folder. To learn about these files, see “Mapping the String Group names to the associated resource files”.
4. In the Solution Explorer, confirm that each file has the property **Build Action** set to **Embedded Resource**.
5. Point DES to your resources.
6. Open the **Global Settings Editor**. It is available in the Start menu or in the [DES Product folder].
7. Select the **custom.DES.config** file in [webapplicationroot]\DES using the  button.
8. Select the String Lookup System item from the list. It is at the bottom.
9. Change the **ResAssemblyName** property to have the same name as your web application's assembly, omitting the “.dll”.
10. Save and close the **Global Settings Editor**. *You may need to restart Visual Studio and/or your web application to see changes made in the custom.des.config file.*

Non-Visual Studio.Net Users

See these topics of the .net documentation:

Creating Resource Files

Creating and Editing Resource Files with ResEditor

Supporting Cultures

If you are supporting cultures and localization, start by determining the cultures that you want to support. See [http://msdn2.microsoft.com/en-us/library/87k6sx8t\(vs.71\).aspx](http://msdn2.microsoft.com/en-us/library/87k6sx8t(vs.71).aspx) for a list of culture names.

The Resource Manager provides a fallback process where it looks for an exact match to the culture name, then to the language, then to the default.

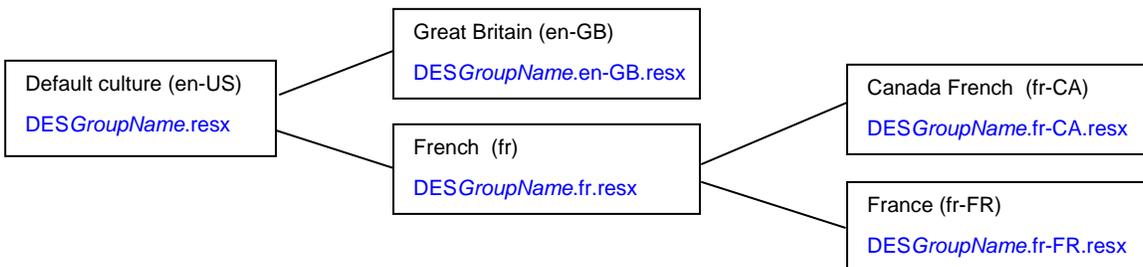
Your strategy is to create several .resx files. Each will have a common “base name”. All except the default type will have the language or language-region in the name in this style:

<basename>.<language>.resx (“DESGroupName.en.resx”)

<basename>.<language>-<region>.resx (“DESGroupName.en-US.resx”)

Example

Supposed your Default culture file contains English for the United States. You want to support English in Great Britain, French for Canada and French for France. Here is how you might define the files:



Entering Strings and Compiling Them

Visual Studio Users

To edit, open the .resx file from the Solution Explorer. Visual Studio provides a row-column interface where each row represents one record.

Start in a blank row. Enter the **LookupID** into the **Name** column and the localized text into the **Value** column. Create one row for each LookupID. *The LookupID is the value entered into a control's property that ends with "LookupID".*

When done save the file.

To compile, build your web application project. Visual Studio automatically compiles all .resx files for you.

Non-Visual Studio.Net Users

1. Edit the .resx file. The .resx file is an XML file that you can edit in Notepad or your favorite editor. See these topics in the .net documentation.

Resources in .Resx File Format

Creating and Editing Resource Files with ResEditor

2. Compile the .resx file into a .resources file. Be sure to include the name of the assembly in the file name like this:

```
[assemblyname].[resxfilename].resources
```

For example, if your assembly is called "MyResources" and you are compiling the DESErrorMessage.resx file, it should be named "MyResources.DESErrorMessage.resources".

Note: If you want to use another naming convention, you must load the resource files yourself in the Application_Start() method. DES's loader uses the above format. See "Properties of the PeterBlum.DES.StringLookup Class".

See this topic in the .net documentation:

Resource File Generator (ResGen.exe)

Examples of using RegGen.exe:

```
resgen.exe DESErrorMessage.resx MyResources.DESErrorMessage.resources
resgen.exe DESLabels.resx MyResources.DESLabels.resources
resgen.exe DESValidationMisc.resx MyResources.DESValidationMisc.resources
```

3. Use the AL.exe application to place the .resources file into an assembly. *DES can only read its resources from within assemblies.* See this topic in the .net documentation:

Creating Satellite Assemblies

Examples of using AL.exe to create "MyResources.dll":

```
al /t:lib
  /embed:MyResources.DESErrorMessage.resources
  /embed:MyResources.DESLabels.resources
  /embed:MyResources.DESValidationMisc.resources
  /out:MyResources.dll
```

4. If necessary, copy the assembly into your web application's **bin** folder.

Using non-default Resource files and Assemblies

Note: If you followed the steps in “Adding These Files In Visual Studio 2005/2008” or “Adding These Files In Visual Studio 2002-2003 and Web Application Projects in VS2005/8”, you may have already done this. Let this section serve as a reference.

DES provides default resource file names. To learn about these files, see “Mapping the String Group names to the associated resource files”. If you want to use alternative names, such as other resource files that you already have, use this section.

DES needs to know the following in order to use your resources:

- The name of the assembly containing the resources or an Assembly object referencing the assembly.
- The base names of the resource files for each group (messages, labels, confirm, etc.) or ResourceManager objects for each group.

DES maintains these settings in the class `PeterBlum.DES.StringLookup`. You can set them up in two places:

- **custom.des.config** – Run the **Global Settings Editor** and edit the section String Lookup System.
- **Global.asax** – The `Application_Start()` method can set up any of the properties on `PeterBlum.DES.StringLookup`. It is generally used to create an Assembly object and ResourceManager objects when the Assembly is not in the `\bin` folder.

Properties of the `PeterBlum.DES.StringLookup` Class

The properties of `PeterBlum.DES.StringLookup` are static/shared. You can set them in the `Application_Start()` method or with the **Global Settings Editor**.

- **ResAssemblyName** (string) – The name of the assembly containing all of the resources. This name should not include the `.dll` extension or the file path. The file must be in the `\bin` folder of the web application.

If you are using Visual Studio 2005 or 2008 (except when using Web Application Projects), leave it unassigned.

If you are using Visual Studio 2002 or 2003 or Web Application Projects in VS2005+, use the name of your web application assembly.

If you are not using Visual Studio, you may have several assemblies ending in “.resource.dll”. There should also be one without “.resource”. That one should be used.

Alternatively, you can set the **ResourceAssembly** property in the `Application_Start()` method.

- **ResourceAssembly** (System.Reflection.Assembly) – The Assembly object that references the assembly containing the resources. This is set automatically when you use **ResAssemblyName**. You should set it when **ResAssemblyName** is not enough to locate the assembly file. For example, it is not located in the `\bin` folder. You set this programmatically in the `Application_Start()` method.
- **ErrorMessageResourceManager** (System.Resources.ResourceManager) – The ResourceManager object that refers to the resources for the ErrorMessage String Group.

Example

Assign programmatically in `Application_Start()` like this:

```
PeterBlum.DES.StringLookup.ErrorMessageResourceManager =
    new ResourceManager("DESErrorMessages",
        System.Reflection.Assembly.GetExecutingAssembly())
```

Using Global Settings Editor

In the String Lookup System section, set the **ErrorMessageResXFile** property to the base filename without any file extension. (“DESErrorMessages”, not “DESErrorMessages.en-GB.resx”)

- **LabelsResourceManager** (System.Resources.ResourceManager) – The ResourceManager object that refers to the resources for the Labels String Group.

Example

Assign programmatically in `Application_Start()` like this:

```
PeterBlum.DES.StringLookup.LabelsResourceManager =
    new ResourceManager("DESLabels",
        System.Reflection.Assembly.GetExecutingAssembly())
```

Using Global Settings Editor

In the String Lookup System section, set the **LabelResXFile** property to the base filename without any file extension. (“DESLabels”, not “DESLabels.en-GB.resx”)

- **ValidationMiscResourceManager** (System.Resources.ResourceManager) – The ResourceManager object that refers to the resources for the ValidationMisc String Group.

Example

Assign programmatically in `Application_Start()` like this:

```
PeterBlum.DES.StringLookup.ValidationMiscResourceManager =
    new ResourceManager("DESValidationMisc",
        System.Reflection.Assembly.GetExecutingAssembly())
```

Using Global Settings Editor

In the String Lookup System section, set the **ValMiscResXFile** property to the base filename without any file extension. (“DESValidationMisc”, not “DESValidationMisc.en-GB.resx”)

- **DateResourceManager** (System.Resources.ResourceManager) – The ResourceManager object that refers to the resources for the DateTime String Group.

Example

Assign programmatically in `Application_Start()` like this:

```
PeterBlum.DES.StringLookup.DateResourceManager =
    new ResourceManager("DESDateTime",
        System.Reflection.Assembly.GetExecutingAssembly())
```

Using Global Settings Editor

In the String Lookup System section, set the **DateTimeResXFile** property to the base filename without any file extension. (“DESDateTime”, not “DESDateTime.en-GB.resx”)

- **HintResourceManager** (System.Resources.ResourceManager) – The ResourceManager object that refers to the resources for the Hint String Group.

Example

Assign programmatically in `Application_Start()` like this:

```
PeterBlum.DES.StringLookup.HintResourceManager =
    new ResourceManager("DESHints",
        System.Reflection.Assembly.GetExecutingAssembly())
```

Using Global Settings Editor

In the String Lookup System section, set the **HintResXFile** property to the base filename without any file extension. (“DESHints”, not “DESHints.en-GB.resx”)

- **ConfirmResourceManager** (System.Resources.ResourceManager) – The ResourceManager object that refers to the resources for the Confirm String Group.

Example

Assign programmatically in `Application_Start()` like this:

```
PeterBlum.DES.StringLookup.ConfirmResourceManager =
    new ResourceManager("DESConfirmMessages",
        System.Reflection.Assembly.GetExecutingAssembly())
```

Using Global Settings Editor

In the String Lookup System section, set the **ConfirmResXFile** property to the base filename without any file extension. (“DESConfirmMessages”, not “DESConfirmMessages.en-GB.resx”)

- **PopupViewResourceManager** (System.Resources.ResourceManager) – The ResourceManager object that refers to the resources for the PopupView String Group.

Example

Assign programmatically in `Application_Start()` like this:

```
PeterBlum.DES.StringLookup.PopupViewResourceManager =
    new ResourceManager("DESPopupViews",
        System.Reflection.Assembly.GetExecutingAssembly())
```

Using Global Settings Editor

In the String Lookup System section, set the **PopupViewResXFile** property to the base filename without any file extension. (“DESPopupViews”, not “DESPopupViews.en-GB.resx”)

- **ContextMenuResourceManager** (System.Resources.ResourceManager) – The ResourceManager object that refers to the resources for the ContextMenu String Group.

Example

Assign programmatically in `Application_Start()` like this:

```
PeterBlum.DES.StringLookup.ContextMenuResourceManager =
    new ResourceManager("DESContextMenu",
        System.Reflection.Assembly.GetExecutingAssembly())
```

Using Global Settings Editor

In the String Lookup System section, set the **ContextMenuResXFile** property to the base filename without any file extension. (“DESContextMenu”, not “DESContextMenu.en-GB.resx”)

- **TextCounterResourceManager** (System.Resources.ResourceManager) – The ResourceManager object that refers to the resources for the TextCounter String Group.

Example

Assign programmatically in `Application_Start()` like this:

```
PeterBlum.DES.StringLookup.TextCounterResourceManager =
    new ResourceManager("DESTextCounter",
        System.Reflection.Assembly.GetExecutingAssembly())
```

Using Global Settings Editor

In the String Lookup System section, set the **TextCounterResXFile** property to the base filename without any file extension. (“DESTextCounter”, not “DESTextCounter.en-GB.resx”)

- **MiscResourceManager** (System.Resources.ResourceManager) – The ResourceManager object that refers to the resources for the Misc String Group.

Example

Assign programmatically in `Application_Start()` like this:

```
PeterBlum.DES.StringLookup.MiscResourceManager =  
    new ResourceManager("DESMisc",  
        System.Reflection.Assembly.GetExecutingAssembly())
```

Using Global Settings Editor

In the String Lookup System section, set the **MiscResXFile** property to the base filename without any file extension. (“DESMisc”, not “DESMisc.en-GB.resx”)

Using a Database

DES lets you use a database as a datasource. You can define the structure of your tables in any way you choose. Perhaps you have a third party localization program that uses your database. To access the database, DES requires that you add a stored procedure called "DESLookupString" to your database. In addition, DES has several global properties in its PeterBlum.DES.StringLookup class that determine the database connection string.

There are three steps to using a database for localization:

1. Set up a database with the appropriate tables.
2. Add the DESLookupString stored procedure.
3. Configure DES with a database connection string.

Click on any of these topics to jump to them:

- ◆ [Mapping String Group Names to their Associated Values in the Table](#)
- ◆ [Set up a Database With The Appropriate Tables](#)
- ◆ [Add the DESLookupString Stored Procedure](#)
- ◆ [Provide the ConnectionString to the Database](#)

Mapping String Group Names to their Associated Values in the Table

This section helps you associate the String Group with the associated name used in the StringGroup column in your table of strings.

String Group Name	Column Value	Usage
ErrorMessages	“ERRORMSG”	Validator.ErrorMessage and Validator.SummaryErrorMessage properties. Also includes PopupErrorFormatter.ErrorMessageHelpLookupID.
Labels	“LABEL”	LocalizableLabel.TextLookupID and the Validator.Label.TextLookupID properties.
ValidationMisc	“VALMISC”	All properties on ValidationSummary, NoErrorFormatter, RequiredFieldMarker, and RequiredFieldDescriptors. Properties on ErrorFormatters except for tooltips and hints.
DateTime	“DATETIME”	All Date and Time module properties except HintLookupID, HintHelpLookupID and TooltipLookupID on the TextBoxes, which are in the Hints Group.
ConfirmMessages	“CONFIRM”	For confirm message boxes.
PopupViews	“POPUVIEW”	For elements in the PopupView system that are not the actual message. (Messages come from ErrorMessages, Hints, and ToolTips)
Hints	“HINT”	For HintLookupID, HintHelpLookupID, and ToolTipLookupID properties on various textboxes, buttons, and the NativeControlExtender. It also handles hints added to ContextMenus.
ContextMenus	“MENU”	ContextMenu properties, except the ConfirmMessageLookupID which is in the ConfirmMessages Group.
TextCounter	“COUNTER”	TextCounter – TextCounter control properties
Misc	“MISC”	For strings not handled by any DES controls. The user can call the String Lookup System to request strings from this file and assign the results to other controls

Set up a Database With The Appropriate Tables

You can construct tables for string lookup any way you want. DES will supply you with an RFC 1766 standard for the culture name and a string group. You can have columns for these or map them to separate tables. See “Elements Needed In Your DataSource” for details. Your DESLookupString stored procedure will resolve the parameters supplied by DES.

If you do not have any tables set up yet, DES includes a SQL script that installs a table and compatible DESLookupString stored procedure. Look in **[DES Installation Folder]\StringLookup\DatabaseStringLookup.sql**. Review this file to see if this will work for you.

Add the DESLookupString Stored Procedure

You must add a stored procedure named “DESLookupString” to your database. The function is available in this sql file: **[DES Product Folder]\StringLookup\DatabaseStringLookup.sql**

Here are the parameters:

- CultureName varchar (5) – The culture name. It is always the RFC 1766 standard, with 5 characters. Your code should attempt to match to this name exactly. If it fails, it should match to the language part. If that fails, it should look in the default culture.
- StringGroup varchar (10) – A string representation of the String Group. Here are the values it will get: “ERRORMSG”, “LABEL”, “VALMISC”, “MENU”, “DATETIME”, “HINT”, “CONFIRM”, “POPUVIEW”, “COUNTER”, “MISC”.
For details, see “Mapping String Group Names to their Associated Values in the Table”.
- LookupID varchar (50) – The value from the LookupID property.
- LocalizedText varchar (4000) – OUTPUT PARAMETER. This is the localized string found. If there was no match, set it to null.

DESLookupString() must return 1 for found and 0 for not found.

Here is a framework for DESLookupString():

```
CREATE PROCEDURE DESLookupString
  @CultureName  varchar(5),
  @StrGroup    varchar(10),
  @LookupID    nvarchar (50),
  @LocalizedText  nvarchar (4000) OUTPUT
AS

SET @LocalizedText = NULL
SET @StrGroup = RTRIM(@StrGroup)
DECLARE @LangName varchar(2) -- language part of @CultureName
SET @LangName = LEFT(@CultureName, 2)

-- Find @LocalizedText by matching to @CultureName.
-- If not found, match to @LangName
-- If not found, use the Default culture ("")

-- Return 1 for found; 0 for not found
IF (@LocalizedText IS NULL)
BEGIN
  RETURN 0
END
ELSE
BEGIN
  RETURN 1
END
END
```

Provide the ConnectionString to the Database

DES needs a `ConnectionString` to your database. It is established either in the **Global Settings Editor** within the String Lookup Section or programmatically in `Application_Start()` by editing the properties shown below. When using `Application_Start()`, these properties are static/shared on the `PeterBlum.DES.StringLookup` class.

*Note: Design mode will lookup text from your database once this is set up. If your development computer does not have access to the database through the connection string, errors will show in Design Mode. Run the Global Settings Editor and change the **DesignModeOn** property to *false*.*

- **ConnectionStringName** (string) – Users of ASP.NET 2.0 can establish their connection string within the `<connectionstrings />` section of the **web.config** file. For example:

```
<connectionstrings>
  <add name="NorthWind"
        connectionString="server=.;database=NorthWind;Integrated Security=SSPI"
        providerName="System.Data.SqlClient" />
</connectionstrings>
```

Set the **ConnectionStringName** property using the name attribute from the **web.config** file. For example, "NorthWind".

- **SqlConnectionName** (string) – If you are using the Microsoft SQL Server 7 or higher database, assign a connection string here.
- **OleDbConnectionString** (string) – If you are using a database that supports an OleDb connection string, assign an OleDb compatible connection string here.

Writing Your Own Lookup String Event Handler

If you cannot use the Resource Manager or Database models described above, you can set up the **OnLookupString** property to your own event handler method.

There are two steps to using your own event handler:

1. Create the event handler method. Suggested location: the **Global.asax** file.
2. Assign it to the **OnLookupString** property in the `Application_Start()` method of **Global.asax**.

Create the Event Handler Method

The **OnLookupString** property uses this definition for methods:

[C#]

```
public delegate void LookupStringHandler(LookupStringHandlerArgs args);
```

[VB]

```
Public Delegate Sub LookupStringHandler( _  
    ByVal args As LookupStringHandlerArgs)
```

The `PeterBlum.DES.LookupStringHandlerArgs` class provides two-way communicate with your method. Some properties are inputs. Others are outputs. Generally, you will use the **CultureName**, **StringGroup**, and **LookupID** properties to find the text in your own datasource. If text is found, set **Found** to `true` and **Result** to the text that was found.

Here are the properties of `PeterBlum.DES.LookupStringHandlerArgs`:

- **LookupID** (string) – The value from the **LookupID** property.
- **CultureName** (string) – The culture name from **PeterBlum.DES.Globals.Page.CultureInfo.Name**. It follows the RFC 1766 standard. Your code should attempt to match to this name exactly. If it fails, it should match to the language part. If that fails, it should look in the default culture.
- **Group** (`PeterBlum.DES.StringLookup.StringLookupGroup`) – Identifies the String Group through an enumerated type. The next property, **StringGroup**, provides a similar service, with a string for the group name. The enumerated type `PeterBlum.DES.StringLookup.StringLookupGroup` has these values:
 - `ErrorMessages`
 - `Labels`
 - `ValidationMisc`
 - `DateTime`
 - `ConfirmMessages`
 - `PopupViews`
 - `Hints`
 - `ContextMenus`
 - `TextCounter`
 - `Misc`
- **StringGroup** (string) – Identifies the String Group through a string. It will be passed one of these values, which correspond to the above **Group** property: “`ERRORMSG`”, “`LABEL`”, “`VALMISC`”, “`MENU`”, “`DATETIME`”, “`HINT`”, “`CONFIRM`”, “`POPUPVIEW`”, “`COUNTER`”, “`MISC`”.
- **Default** (string) – The default string associated with the **LookupID**. For example, if the **LookupID** was `Validator.ErrorMessageLookupID`, this is `Validator.ErrorMessage`. In some localization systems, they will ignore the **LookupID** and translate using this value.
- **Found** (Boolean) – Set this to `true` when your event finds a match and returns it. It defaults to `false`.
- **Result** (string) – Set this to the text found. If nothing was found, do nothing with this value.

Example

DES sets the **OnLookupString** property to the `ResourceManagerLookupHandler` method when you use the Resource Manager. Here is a slightly edited version of that method.

[C#]

```
uses PeterBlum.DES;
...
public void ResourceManagerLookupHandler(
    LookupStringHandlerArgs args)
{
    ResourceManager vRM = null;
    switch (args.Group)
    {
        case StringLookup.StringLookupGroup.ErrorMessages:
            vRM = StringLookup.ErrorMessagesResourceManager;
            break;
        case StringLookup.StringLookupGroup.Labels:
            vRM = StringLookup.LabelsResourceManager;
            break;
        case StringLookup.StringLookupGroup.ValidationMisc:
            vRM = StringLookup.ValidationMiscResourceManager;
            break;
    }
    /* There are more groups than shown here */
    // switch

    if (vRM != null)
        try
        {
            args.Result = vRM.GetString(args.LookupID,
                PeterBlum.DES.Globals.Page.CultureInfo);
            args.Found = true;
        }
        catch (Exception)
        {
            // not found. Do nothing.
        }
    // ResourceManagerLookupHandler
}
```

[VB]

```
Imports PeterBlum.DES
...
Public Sub ResourceManagerLookupHandler( _
    ByVal args As LookupStringHandlerArgs)

    Dim vRM As ResourceManager = nothing
    Select args.Group
        Case StringLookup.StringLookupGroup.ErrorMessages
            vRM = StringLookup.ErrorMessagesResourceManager
        Case StringLookup.StringLookupGroup.Labels
            vRM = StringLookup.LabelsResourceManager
        Case StringLookup.StringLookupGroup.ValidationMisc
            vRM = StringLookup.ValidationMiscResourceManager
    ' There are more groups than shown here
    End Select

    If vRM <> Nothing Then
        Try
            args.Result = vRM.GetString(args.LookupID,
                PeterBlum.DES.Globals.Page.CultureInfo)
            args.Found = true
        Catch e As Exception
            ' not found. Do nothing.
        End Try
    End Sub
```

Using the OnLookupString property

The **OnLookupString** property is on the `PeterBlum.DES.StringLookup` class as a static/shared property. Set it in the **Global.asax** file in the `Application_Start()` method.

[C#]

```
protected void Application_Start(Object sender, EventArgs e)
{
    PeterBlum.DES.StringLookup.OnLookupString =
        new PeterBlum.DES.LookupStringHandler(YourEventHandler);
}
```

[VB]

```
Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
    PeterBlum.DES.StringLookup.OnLookupString = _
        New PeterBlum.DES.LookupStringHandler(AddressOf YourEventHandler)
End Sub
```

Calling The String Lookup System

DES's controls automatically use the String Lookup System when they have their LookupID property set up. You can use the String Lookup System with other controls and elements on your page. Simply call the `PeterBlum.DES.StringLookup.StringIDLookup()` method to convert your own LookupID into its looked-up value. `StringIDLookup()` respects the current culture assigned to **PeterBlum.DES.Globals.Page.CultureInfo**.

PeterBlum.DES.StringLookup.StringIDLookup Method

[C#]

```
public static string StringIDLookup(string pLookupID, string pDefault,
    PeterBlum.DES.StringLookup.StringLookupGroup pGroup)
```

[VB]

```
Public Shared Sub StringIDLookup (ByVal pLookupID As String, _
    ByVal pDefault As String,
    ByVal pGroup As PeterBlum.DES.StringLookup.StringLookupGroup)
```

Parameters

pLookupID

This is an identifier used to find a particular string. When using the Resource Manager, this maps to the Name attribute.

pDefault

A string to use when no match was found to *pLookupID*. It can be an empty string.

pGroup

Identifies the String Group through an enumerated type. The enumerated type `PeterBlum.DES.StringLookup.StringLookupGroup` has these values:

- o ErrorMessage
- o Labels
- o ValidationMisc
- o DateTime
- o ConfirmMessages
- o PopupViews
- o Hints
- o ContextMenus
- o TextCounter
- o Misc

Return value

The string requested. It will follow the culture setting in `PeterBlum.DES.Globals.Page.CultureInfo`. If no match was found, the value of the *pDefault* parameter is returned.

This is a static/shared method. You do not create an object before using it. Simply specify the `StringLookup` class like this:

```
result = PeterBlum.DES.StringLookup.StringIDLookup(
    "lookupID", "default", PeterBlum.DES.StringLookup.StringLookupGroup.GroupValue)
```

Example

Add three strings to a DropDownList control. The user has defined these strings in the MISC String Group.

[C#]

```
PeterBlum.DES.StringLookup.StringLookupGroup vGroup =  
    PeterBlum.DES.StringLookup.StringLookupGroup.Misc;  
DropDownList1.Items.Add(  
    PeterBlum.DES.StringLookup.StringLookupID(  
        "USA", "United States", vGroup));  
DropDownList1.Items.Add(  
    PeterBlum.DES.StringLookup.StringLookupID(  
        "Canada", "Canada", vGroup));  
DropDownList1.Items.Add(  
    PeterBlum.DES.StringLookup.StringLookupID(  
        "Other", "Other", vGroup));
```

[VB]

```
Dim vGroup As PeterBlum.DES.StringLookup.StringLookupGroup = _  
    PeterBlum.DES.StringLookup.StringLookupGroup.Misc  
DropDownList1.Items.Add( _  
    PeterBlum.DES.StringLookup.StringLookupID( _  
        "USA", "United States", vGroup))  
DropDownList1.Items.Add( _  
    PeterBlum.DES.StringLookup.StringLookupID( _  
        "Canada", "Canada", vGroup))  
DropDownList1.Items.Add( _  
    PeterBlum.DES.StringLookup.StringLookupID( _  
        "Other", "Other", vGroup))
```

The ViewState and Preserving Properties forPostBack

The ViewState is a property on each control and the Page that can preserve the last setting of various properties as the page is recreated on PostBack. The idea is very simple and effective. Its design requires that each preserved property gets converted into a string and written into the page (in the `_ViewState` hidden field). This string can greatly increase the size of the page. If you don't use the ViewState on a control, Microsoft provides the **EnableViewState** property to turn it off. By default, the ViewState is on and it preserves most of the properties that are not at their default state into the page.

Each time the ViewState is assigned after the control is initialized, any data added to the ViewState is preserved. There is no need to preserve most properties because you manage its setting on post back. But the ViewState still does, increasing the size of the page meaninglessly. Microsoft designed it this way because it's easy to use. The real-world usage of ViewState has proven that page size needs to be considered.

DES's controls rarely benefit from the ViewState. So instead of increasing the size of the page, they use a minimal ViewState. For most controls, this comprises of the **Enabled** and **Visible**. See "Properties Automatically Saved in the ViewState". (Controls do store their own private data in the ViewState as needed. You shouldn't try to disable that.)

Use the **ViewStateMgr** property on each DES control to save other properties into the ViewState. **ViewStateMgr** (type `PeterBlum.DES.ViewStateMgr`) supplies properties and methods to manage the ViewState on the control.

TrackProperty Method

Use the `TrackProperty()` method on the **ViewStateMgr** property to remember a property. Generally this is done when you programmatically change the property and expect the same result after a postback or AJAX callback.

Suppose that you want to preserve the **AutoPostBack** property. Call the `TrackProperty()` method passing "AutoPostBack" like this:

```
TextBox1.ViewStateMgr.TrackProperty("AutoPostBack")
```

You can assign most properties by specifying their name. (The name must be a case sensitive match to the property name.) If you want to assign a property that is a child to another property, use this format for the name: "ParentPropertyName.ChildPropertyName". For example:

```
Validator1.ViewStateMgr.TrackProperty("ErrorFormatter.ForeColor")
```

```
Validator1.ViewStateMgr.TrackProperty("Enabler.ControlIDToEvaluate")
```

This feature has its limitations to the properties it can store. It will throw exceptions at runtime when you give it something that it cannot handle. So test your settings.

After post back, DES automatically continues to preserve any property identified in `TrackProperty()`. If you want to remove a property, call the `RemoveProperty()` method, passing the name of the property. For example:

```
TextBox1.ViewStateMgr.RemoveProperty("AutoPostBack")
```

ViewState Property – Promoting the ViewState to Public

Each control has its own **ViewState** property. It would be a great place for you to store something associated with the control for use on post back. For example, if you want to recreate a list of conditions in `CountTrueConditionsValidator`, you might want to store the number of conditions.

Unfortunately, the **ViewState** property is not public. So you cannot access it. The **ViewStateMgr** property provides you with another **ViewState** property. You can store your settings into this one. Otherwise, it works the same as the one on the control itself.

Setting the ViewState:

```
Validator1.ViewStateMgr.ViewState.Add("UniqueName", value)
```

UniqueName is a string that uniquely represents your data within this ViewState collection. *Value* is any serializable data or object.

Getting the ViewState:

```
obj = Validator1.ViewStateMgr.ViewState["UniqueName"]
```

(In VB.Net, replace brackets with parenthesis.)

This will return the value. Be sure to typecast to your value type. If the *UniqueName* is not found, it will return null/nothing.

Properties Automatically Saved in the ViewState

Here are the properties that are automatically preserved in the ViewState.

Controls	Properties
All TextBoxes	Visible, Enabled, ReadOnly, Text.
Calendar, MultiSelectionCalendar	Visible, Enabled, and any other property inherited from the System.Web.UI.WebControls.Panel control. SelectedDate, SelectedDates, TodaysDate, MinDate, MaxDate, and SpecialDate
MonthYearPicker	Visible, Enabled, and any other property inherited from the System.Web.UI.WebControls.Panel control. Month, Year, MinDate, and MaxDate
TimePicker	Visible, Enabled, and any other property inherited from the System.Web.UI.WebControls.Panel control. SelectedTime
PopupMonthYearPicker, PopupCalendar, PopupTimePicker	Visible, Enabled, and any other property inherited from the System.Web.UI.WebControls.WebControl base class.

Page Level Properties and Methods Used by Most Controls

Most of the behavior and formatting of the controls are found in their own properties. DES also provides properties that are not found on any control. They are on the **PeterBlum.DES.Globals.Page** object. **PeterBlum.DES.Globals.Page** offers “page-level” settings, values shared between all controls on a page.

This section covers properties and methods on **PeterBlum.DES.Globals.Page** used throughout the DES controls.

Click on any of these topics to jump to them:

- ◆ What is the PeterBlum.DES.Globals.Page property?
- ◆ Properties on PeterBlum.DES.Globals.Page
 - Debugging PeterBlum.DES.Globals.Page Properties
- ◆ Methods on PeterBlum.DES.Globals.Page
 - AttachCodeToEvent Method

What is the PeterBlum.DES.Globals.Page property?

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.

Properties on PeterBlum.DES.Globals.Page

You generally assign properties to **PeterBlum.DES.Globals.Page** in your `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 or the `<% @PAGE %>` tag with the Culture property.

```
<%@Page Culture="en-US" [other page properties] %>
```

You can set it programmatically in your `Page_Load()` method. 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")
```

The default `CultureInfo` object is read-only. (It is a direct reference to the same object on `CultureInfo.CurrentCulture` which Microsoft makes read-only.) If you want to modify the properties of this object, you must create a new one. Either use the `CreateSpecificCulture()` method or `Clone()` the current one like this:

[C#]

```
PeterBlum.DES.Globals.Page.CultureInfo =
    (CultureInfo) PeterBlum.DES.Globals.Page.CultureInfo.Clone();
PeterBlum.DES.Globals.Page.CultureInfo.property = value;
```

[VB]

```
PeterBlum.DES.Globals.Page.CultureInfo = _
    CType(PeterBlum.DES.Globals.Page.CultureInfo.Clone(), CultureInfo)
PeterBlum.DES.Globals.Page.CultureInfo.property = value
```

- **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. See “The TrueBrowser Class”.
- **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
```

Page.SetFocus()

ASP.NET 2 and higher.

The **InitialFocusControl** property serves the same purpose as the `Page.SetFocus()` method. If you use both, they will each set up their JavaScript and the last one to run its JavaScript will establish focus.

Recommendation: Use the **InitialFocusControl** property.

- **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. 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**.
- **PageIsLoadingMsg** (string) – The error message to display on the client-side if the user interacts with a control before it is initialized. It defaults to "Page is loading. Please wait."
- **IsPostBack** (Boolean) - DES behaves differently depending on the state of the **Page.IsPostBack** property. Sometimes users need to control this behavior. This property reflects the current state of **Page.IsPostBack**. However, the user can change its value, either to `false` (not postback) or to `true` (postback). Here are features that use this property:
 - The **ValidationSummary** control only shows up when this is `true`.
 - The Validator's **NoErrorFormatter.Mode** property.
 - **PeterBlum.DES.Globals.Page.ShowAlertOnSubmit** uses this to build an alert that appears after postback.
 - **PeterBlum.DES.Globals.Page.FocusOnSubmit** can sets the focus after post back.
 - The `<form onreset=>` code varies based on this.
- **Validators** (PeterBlum.DES.ValidatorCollection) – A collection of the Validators on this page. Available for page developers to iterate through as they need to review and modify settings.
- **SetFocusFunctionName** (string) – The name of a client-side function that is called whenever DES attempts to set focus to a field with an error. Use this function when the field is invisible and you want to show it. For example, if your fields are in a tabbed user interface, a field with the error is invisible when on a tabbed panel that is not visible. Your function can entirely replace the built in focus setting code. For example, if the field is a **RichTextBox** and it has a specialized JavaScript function to select the entire text when setting focus, use this.

The function must take one parameter, the element that is getting the focus. It is an object, defined in DOM/DHTML, such as the `<input>` tag for a textbox. One key property of every element is its 'id'. Use that to determine the element so you know what you are attempting to act on.

The function must return a Boolean value where `true` means set focus and `false` means do not set focus. When you replace the focus setting code with your own, return `false`.

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.

Your function should appear on the page that is generated.

In this example, the parent of element id 'TextBox1' may be invisible. This JavaScript code makes it visible and returns true to indicate that it can set focus. **PeterBlum.DES.Globals.Page.SetFocusFunctionName** is set to "ShowTextBox1".

```
function ShowTextBox1(pFld)
{
    if (pFld.id == 'TextBox1')
        pFld.parentNode.style.visibility = 'inherit';
    return true;
}
```

- **EnableButtonImageEffects** (enum `PeterBlum.DES.EnableButtonImageEffects`) – Many buttons can show up to 3 images: normal, pressed, and mouseover. You only need to specify the name of the normal image and provide two more with the same name + “pressed” and “mouseover”. DES will “sniff” your local folder to detect these files. The sniffing cannot see all possible URLs, including those starting with “http://”. Use this property to override the sniffer and specify that the images are present or not. *Alternatively, the URL can be pipe delimited with URLs for normal/pressed/mouseover.*

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

- o None – Never use image effects.
- o Always – Always use image effects. Assume that all image files are available.
- o Auto – Detect the files, if possible before using them.
- o Pressed – Always set up for pressed. Never set up for mouse over.
- o MouseOver – Always set up for mouseover. Never set up for pressed.

It defaults to `EnableButtonImageEffects.Auto`. There is no global setting for this property.

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 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, “myimagepressed.gif”. Create the mouseover version and give it the same name, with “MouseOver” added before the extension. For example, myimagemouseover.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 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, “myimagepressed.gif”. Create the mouseover version and give it the same name, with “MouseOver” added before the extension. For example, myimagemouseover.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 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).

Debugging PeterBlum.DES.Globals.Page Properties

If you are uncertain that the values set in **PeterBlum.DES.Globals.Page** are correct, use the `PeterBlum.DES.Globals.Page.DescribeProperties()` method. It returns a string that you can assign to a `Label` or `LiteralControl` control to show on the page or you can set up `<@ Page Trace="true" >` and output them using `Page.Trace.Write()`.

```
PeterBlum.DES.Globals.Page.DescribeProperties(showHTML)
```

showHTML

Pass `true` to format the text in HTML format (as a table) and `false` to format it as a carriage return delimited set of lines useful to output to a file or other system that cannot use HTML.

Call it after you have set any properties on **PeterBlum.DES.Globals.Page**, like this.

```
DebugLabel1.Text = PeterBlum.DES.Globals.Page.DescribeProperties(true)
Page.Trace.Write(PeterBlum.DES.Globals.Page.DescribeProperties(false))
```

Methods on PeterBlum.DES.Globals.Page

These methods can assist you in page development.

AttachCodeToEvent Method

Adds JavaScript to a client-side event on a control. This is a replacement for this type of code:

`Control.Attributes.Add("oneventname", "your code")`. It builds a wrapper around your code that safely merges your code with any code that is already assigned to that event, either before or after the existing code. For example, the `IntegerTextBox` is already using the `onfocus` event. Use this to combine your own `onfocus` code with that found in the `IntegerTextBox`.

[C#]

```
public void AttachCodeToEvent(Control pControl, string pEventName,
    string pCode, bool pFirst)
```

[VB]

```
Public Sub AttachCodeToEvent(ByVal pControl As Control,
    ByVal pEventName As String, ByVal pCode As String,
    ByVal pFirst As Boolean)
```

Parameters

pControl

A reference to the control that gets the event.

pEventName

A string that holds the name of the event. It is case sensitive and must match real event names specified in DHTML and DOM or it will not be called by the browser.

Most common events on controls: `onfocus`, `onblur`, `onclick`, `onchange`, `onkeypress`, `onkeydown`.

pCode

Your JavaScript code. Be sure that it is completely valid code or you will encounter JavaScript errors as the page is loaded. Since it may be attached to other JavaScript code, it should end in a valid statement terminator, which is either a semicolon (;) or closing bracket (}).

pFirst

When `true`, your code executes before any code already assigned to the event. When `false`, it executes after any previously assigned code.

This event never assigns your code directly to the `Control.Attributes` collection. Instead, it creates client-side code that runs when the page is loaded, updating anything already on the event. This avoids conflicts between two users of the same attribute value. (For example, the `IntegerTextBox` does this: `Attributes.Add("onkeypress", "[code]")` in the `prerender` event, overwriting anything you may have assigned to it.)

Example

Add some code to the `onblur` event of an `IntegerTextBox`. While normally you would write `IntegerTextBox1.Attributes.Add("onblur", "[your code]")`, to avoid conflicts, you use `PeterBlum.DES.Globals.Page.AttachCodeToEvent()` like this.

[C#]

```
PeterBlum.DES.Globals.Page.AttachCodeToEvent(IntegerTextBox1,
    "onblur", "[your code]", true);
```

[VB]

```
PeterBlum.DES.Globals.Page.AttachCodeToEvent(IntegerTextBox1, _
    "onblur", "[your code]", True)
```

Establishing Localization for the Web Form

ASP.NET provides extensive details on how dates, times, and numbers can be localized within the `CultureInfo` object. Every web form automatically has a `CultureInfo` object loaded into the current thread's `System.Globalization.CultureInfo.CurrentCulture` object and all Peter's Data Entry Suite controls use it through the `PeterBlum.DES.Globals.Page.CultureInfo` property. This makes it easy to define a culture across the page.

Here are several ways to establish the current culture in the `PeterBlum.DES.Globals.Page.CultureInfo` property.

Click on any of these topics to jump to them:

- ◆ Localization for the Entire Web Site
- ◆ Default Localization for a Web Form
- ◆ Change Localization Based on the User's Culture
 - For the Entire Site
 - For the Current Page

Localization for the Entire Web Site

You can set the default `CultureInfo` object in the `web.config` file using this syntax:

```
<configuration>
  <system.web>
    <globalization culture="id-ID" />
  </system.web>
</configuration>
```

Determine the value for `id-ID` from the list shown on the `CultureInfo` overview topic in the .net docs.

Default Localization for a Web Form

You can establish a `CultureInfo` object that affects one page within the `<% @Page %>` tag. Use this technique to establish a default value. You can override it if users have different cultures by using the techniques described in the next section.

```
<% @Page culture="id-ID" %>
```

Determine the value for `id-ID` from the list shown on the `CultureInfo` overview topic in the .net docs.

Change Localization Based on the User's Culture

When your site supports different cultures, you will create some technique to recognize the user's culture. Perhaps when they visit, you ask for their culture. You can get their preferred culture identifier from their browser by checking the **Request.UserLanguages** property, although this requires the user to set up their browser and many users do not do this.

There are two ways to establish the user's culture:

Click on any of these topics to jump to them:

- ◆ For the Entire Site
- ◆ For the Current Page

For the Entire Site

The **Global.asax** file provides the method `Application_BeginRequest()` to execute code that runs whenever any web form is opened. Use it to retrieve the desired `CultureInfo` object and assign it to the current thread's **CurrentCulture** property.

[C#]

```
protected void Application_BeginRequest(object sender, EventArgs e)
{
    string vCultureID = code that determines the ID;
    System.Threading.Thread.CurrentThread.CurrentCulture =
        System.Globalization.CultureInfo.CreateSpecificCulture(vCultureID);
}
```

[VB]

```
Protected Sub Application_BeginRequest(ByVal sender As Object, _
    ByVal e As EventArgs)
    Dim vCultureID As string = code that determines the ID
    System.Threading.Thread.CurrentThread.CurrentCulture = _
        System.Globalization.CultureInfo.CreateSpecificCulture(vCultureID)
End Sub
```

If you want to customize the `CultureInfo` object, you must clone the object retrieved by `CreateSpecificCulture()` because Microsoft has made the object return read-only. They provided a `Clone()` method to help.

[C#]

```
using System.Globalization;
...
protected void Application_BeginRequest(object sender, EventArgs e)
{
    string vCultureID = code that determines the ID;
    CultureInfo vCultureInfo = (CultureInfo)
        CultureInfo.CreateSpecificCulture(vCultureID).Clone();
    vCultureInfo.DateTimeFormat.ShortDatePattern = "MM-dd-yyyy";
    vCultureInfo.DateTimeFormat.DateSeparator = "-";
    System.Threading.Thread.CurrentThread.CurrentCulture = vCultureInfo;
}
```

[VB]

```
Imports System.Globalization
...
Protected Sub Application_BeginRequest(ByVal sender As Object, _
    ByVal e As EventArgs)
    Dim vCultureID As string = code that determines the ID
    Dim vCultureInfo As CultureInfo = _
        CType(CultureInfo.CreateSpecificCulture(vCultureID).Clone(), _
            CultureInfo)
    vCultureInfo.DateTimeFormat.ShortDatePattern = "MM-dd-yyyy"
    vCultureInfo.DateTimeFormat.DateSeparator = "-"
    System.Threading.Thread.CurrentThread.CurrentCulture = vCultureInfo
End Sub
```

For the Current Page

Use the `Page_Init()` method to retrieve the desired `CultureInfo` object and assign it to the `PeterBlum.DES.Globals.Page.CultureInfo` property.

Alert: Always set up the culture prior to getting or setting any value that uses culture on these controls.

[C#]

```
protected void Page_Init(object sender, EventArgs e)
{
    string vCultureID = code that determines the ID;
    PeterBlum.DES.Globals.Page.CultureInfo =
        System.Globalization.CultureInfo.CreateSpecificCulture(vCultureID);
}
```

[VB]

```
Protected Sub Page_Init(ByVal sender As Object, _
    ByVal e As EventArgs)
    Dim vCultureID As string = code that determines the ID
    PeterBlum.DES.Globals.Page.CultureInfo = _
        System.Globalization.CultureInfo.CreateSpecificCulture(vCultureID)
End Sub
```

If you want to customize the `CultureInfo` object, you must clone the object retrieved by `CreateSpecificCulture()` because Microsoft has made the object return read-only. They provided a `Clone()` method to help.

[C#]

```
using System.Globalization;
...
protected void Page_Init(object sender, EventArgs e)
{
    string vCultureID = code that determines the ID;
    vCultureInfo = (CultureInfo)CultureInfo.CreateSpecificCulture(vCultureID).Clone();
    vCultureInfo.DateTimeFormat.ShortDatePattern = "MM-dd-yyyy";
    vCultureInfo.DateTimeFormat.DateSeparator = "-";
    PeterBlum.DES.Globals.Page.CultureInfo = vCultureInfo;
}
```

[VB]

```
Imports System.Globalization
...
Protected Sub Page_Init(ByVal sender As Object, _
    ByVal e As EventArgs)
    Dim vCultureID As string = code that determines the ID
    Dim vCultureInfo As CultureInfo = _
        CType(CultureInfo.CreateSpecificCulture(vCultureID).Clone(), CultureInfo)
    vCultureInfo.DateTimeFormat.ShortDatePattern = "MM-dd-yyyy"
    vCultureInfo.DateTimeFormat.DateSeparator = "-"
    PeterBlum.DES.Globals.Page.CultureInfo = vCultureInfo
End Sub
```

Using Style Sheets

Each of these controls utilizes cascading style sheet definitions for their appearance. Peter's Data Entry Suite provides many style sheet files that provide the default appearance. You edit these style sheet files to customize the appearance. The files are heavily documented to assist you.

There are separate style sheet files for individual types of controls, such as **Calendar.css** for the Calendar control and **Validation.css** for Validator controls. See "Identifying the Style Sheet File for a Specific Control".

Each page's <head> tag needs <link> tag to the style sheet files used by the Peter's Data Entry Suite controls. With little (ASP.NET 1.x) or no (ASP.NET 2.0) set up on each page, DES will insert the <link> tags for you. See "Adding Style Sheet Files To The Page". When it sets them up, it uses a default URL to each file which can be overridden by settings in your **web.config** file and in `Page_Load()`. See "Customizing the URLs to Each Style Sheet File".

While there are many style sheet files, DES is very smart about their usage. It gets only the files needed by the controls on the page. Then it compresses them and combines them into a single <link> tag to minimize their impact on the page's load time. See "Compressing and Merging Files". This compression phase also supports the ASP.NET URL token "~/" and "{APPEARANCE}" token so you can use the styles `background: url(~/folder/filename.gif)` and `background: url({APPEARANCE}folder/filename.gif)`.

Since different browsers sometimes need different style sheet definitions, you can have DES automatically swap a style sheet class name at runtime based on the browser. See "Browser Sensitive Style Sheet Class Names".

Click on any of these topics to jump to them:

- ◆ Adding Style Sheet Files To The Page
 - ASP.NET 2 and above Users
 - ASP.NET 1.x Users
- ◆ Identifying the Style Sheet File for a Specific Control
- ◆ Customizing the URLs to Each Style Sheet File
 - Changing the URLs globally
 - Changing the URL on a Page
 - Disabling the Output of Link Tags Globally
 - Disabling the Output of Link Tags on a Page
- ◆ Browser Sensitive Style Sheet Class Names
- ◆ Compressing and Merging Files
 - Troubleshooting: How to see what DES output
 - Troubleshooting: Changing the URL to GetFiles.aspx
- ◆ Special Parsing Features
- ◆ Support for Your Own Style Sheet Files

Adding Style Sheet Files To The Page

HTML requires the `<link>` tag in the `<head>` section of your web form to attach each style sheet file:

```
<link href="[URL to your css file]" type="text/css" rel="stylesheet" />
```

DES creates this `<link>` tag for you. It also merges and compresses all requested style sheet files into a single `<link>` tag. See “Compressing and Merging Files” for details.

Hint: If any popup appears transparent or a control is poorly formatted, you have not established the style sheets properly.

ASP.NET 2 and above Users

Any web form with the `runat=server` property in the `<head>` tag will automatically add the `<link>` tags to the appropriate style sheet files.

Troubleshooting

If you do not see the affects of the style sheet files:

- Confirm that you have `<head runat="server">` instead of `<head>`
- View the HTML output of the page. It should have a `<link>` tag identifying **GetFiles.aspx** or **DESGetFiles.aspx**, like in this example:

```
<link href="/WebSite1/DES/GetFiles.aspx?type=styles&more parameters"
rel="stylesheet" type="text/css" />
```

or

```
<link href="DESGetFiles.aspx?type=styles&more parameters"
rel="stylesheet" type="text/css" />
```

See “Compressing and Merging Files” for configuration issues with **GetFiles.aspx**.

- If you do not see the `<link>` tag, add this line into the `<head>` tag’s inner HTML:

```
<%= PeterBlum.DES.StyleSheetManager.GetLinkTags() %>
```

See the next section for an example.

Typically this is needed when the `<head>` tag contains the `<% %>` tags. This is a limitation of the `HtmlForm` object that prevents adding child controls to it. When you are not seeing the `<link>` tag in the HTML output, set `<@ Page trace="true" >`. When you run the page, the trace will describe what happened when it attempted to add the `<link>` tags.

- DES can output the contents returned by **GetFiles.aspx** to a page for your review. See “Troubleshooting: How to see what DES output”. If any of the files contents are missing, most likely the URL DES is using to retrieve the file does not point to a file. Correct the location of the file or specify a different URL. See “Customizing the URLs to Each Style Sheet File”.

ASP.NET 1.x Users

DES provides the `PeterBlum.DES.StyleSheetManager.GetLinkTags()` method to return the exact `<link>` tag text to your page. Call it within the `<head>` tag as shown here:

```
<html>
  <head>
    <title>WebForm1</title>
    [various meta tags]
    <%= PeterBlum.DES.StyleSheetManager.GetLinkTags() %>
  </head>
```

Troubleshooting

If you do not see the affects of the style sheet files:

- View the HTML output of the page. It should have a `<link>` tag identifying **GetFiles.aspx** or **DESGetFiles.aspx**, like in this example:

```
<link href="/WebSite1/DES/GetFiles.aspx?type=styles&more parameters"
rel="stylesheet" type="text/css" />
```

or

```
<link href="DESGetFiles.aspx?type=styles&more parameters"
rel="stylesheet" type="text/css" />
```

See “Compressing and Merging Files” for configuration issues with **GetFiles.aspx**.

- DES can output the contents returned by **GetFiles.aspx** to a page for your review. See “Troubleshooting: How to see what DES output”. If any of the files contents are missing, most likely the URL DES is using to retrieve the file does not point to a file. Correct the location of the file or specify a different URL. See “Customizing the URLs to Each Style Sheet File”.

Identifying the Style Sheet File for a Specific Control

DES comes with the following style sheet files in subfolders of the **DES\Appearance** folder in your web application. The subfolders are by product module. These files are heavily documented to assist you in their editing.

Folder/Filename	Controls Supported
Date And Time/Calendar.css	Calendar, PopupCalendar, MultiSelectionCalendar. The popup calendars within the DateTextBox and AnniversaryTextBox
Date And Time/ DateAndTimeTextBoxes.css	DateTextBox, MonthYearTextBox, and AnniversaryTextBox.
Date And Time/ MonthYearPicker.css	MonthYearPicker and PopupMonthYearPicker. The popup MonthYearPickers within Calendars and MonthYearTextBox.
Date And Time/ MultiSelectionCalendar.css	MultiSelectionCalendar
Date And Time/SpecialDates.css	Calendar and MultiSelectionCalendar when using the SpecialDates control
Date And Time/TimePicker.css	TimePicker and PopupTimePicker. The popup TimePicker within the TimeOfDayTextBox and DurationTextBox.
Interactive Pages/ InteractivePages.css	CalculationController, FieldStateController, MultiFieldStateController, FSCOnCommand, and MultiFSCOnCommand.
Interactive Pages/Menu.css	ContextMenu including those within other DES controls
Interactive Pages/PopupHints.css	The PopupHints feature used by HintFormatters
Interactive Pages/TextCounter.css	TextCounter
TextBoxes/TextBoxes.css	TextBox, FilteredTextBox, IntegerTextBox, DecimalTextBox, CurrencyTextBox, PercentTextBox, MultiSegmentDataEntry
Validation/ PopupErrorFormatter.css	The PopupHints feature used by PopupErrorFormatters
Validation/Validation.css	All Validators, ValidationSummary, RequiredFieldMarker, and RequiredFieldsDescription.

You can customize these files as needed. The Peter's Data Entry Suite controls have numerous properties that refer to style sheet class names. They all end in "CssClass". The documentation will show you the default name, which refers to a name found in each of these three files. You can edit these files directly or create new styles in separate style sheet files, so long as the CssClass property refers to a name from one of the <link> or <style> tags on the page.

Customizing the URLs to Each Style Sheet File

You can relocate the style sheet files or use different files. This section describes how to change the URL to a file globally and on a page level.

Click on any of these topics to jump to them:

- ◆ Changing the URLs globally
- ◆ Changing the URL on a Page
- ◆ Disabling the Output of Link Tags Globally
- ◆ Disabling the Output of Link Tags on a Page

Changing the URLs globally

Use the **web.config** file or `Application_Start()` method to define the URLs of each files. You can change one, a few or all, as needed. These settings will affect all pages in your web application, except when overridden on the page (see the next topic).

Note: If you have moved the entire Appearance folder, you can set the `DES_AppearanceVirtualPath` key in `web.config` and it will adjust the location of all style sheet files it contains. You do not need to use the instructions here. See the Installation Guide for details on `DES_AppearanceVirtualPath`.

Using the web.config file

Use these keys in the `<appSettings>` section of **web.config**. Only use keys for the controls you have alternative files. Any that continue to use the default can be omitted.

```
<add key="DES_StyleSheetCalendar" value="[url]/[filename].css" />
<add key="DES_StyleSheetDateAndTimeTextBoxes" value="[url]/[filename].css" />
<add key="DES_StyleSheetMonthYearPicker" value="[url]/[filename].css" />
<add key="DES_StyleSheetMultiSelectionCalendar" value="[url]/[filename].css" />
<add key="DES_StyleSheetSpecialDates" value="[url]/[filename].css" />
<add key="DES_StyleSheetTimePicker" value="[url]/[filename].css" />
<add key="DES_StyleSheetInteractivePages" value="[url]/[filename].css" />
<add key="DES_StyleSheetMenu" value="[url]/[filename].css" />
<add key="DES_StyleSheetPopupHints" value="[url]/[filename].css" />
<add key="DES_StyleSheetTextCounter" value="[url]/[filename].css" />
<add key="DES_StyleSheetTextBoxes" value="[url]/[filename].css" />
<add key="DES_StyleSheetPopupErrorFormatter" value="[url]/[filename].css" />
<add key="DES_StyleSheetValidation" value="[url]/[filename].css" />
```

Note: The .Net framework uses the tilde (~) character to indicate the web application root path. For any files within the application folder, it's recommended that you start the URL with this symbol. For example, "~/StyleSheets/Calendar.css".

WARNING: Do not use URLs that start with `http://` or `https://` unless you turn off the merge and compression feature. See "Modifying the merging and compression features".

Programmatically within Application_Start()

Use the method `PeterBlum.DES.StyleSheetManager.SetDefaultUrl()` to specify a different URL.

[C#]

```
void SetDefaultUrl(PeterBlum.DES.StyleSheetFiles pStyleSheetFile, string pUrl)
```

[VB]

```
Sub SetDefaultUrl(ByVal pStyleSheetFile As PeterBlum.DES.StyleSheetFiles, _
    ByVal pUrl As String)
```

Parameters

pStyleSheetFile

Specifies the style sheet file's usage (association with controls). The enumerated type `PeterBlum.DES.StyleSheetFiles` has these values:

- o Calendar
- o DateAndTime
- o InteractivePages
- o Menu
- o MonthYearPicker
- o MultiSelectionCalendar
- o PopupErrorFormatter
- o PopupHints
- o SpecialDates
- o TextBoxes
- o TextCounter
- o TimePicker
- o Validation
- o VAMCompatible

pUrl

The URL used by the browser to access the file. It can use the ~ character for the web application root folder.

Example

[C#]

```
protected void Application_Start(Object sender, EventArgs e)
{
    PeterBlum.DES.StyleSheetManager.SetDefaultUrl(
        PeterBlum.DES.StyleSheetFiles.Calendar, "~/StyleSheets/Calendar.css");
}
```

[VB]

```
Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
    PeterBlum.DES.StyleSheetManager.SetDefaultUrl( _
        PeterBlum.DES.StyleSheetFiles.Calendar, "~/StyleSheets/Calendar.css")
End Sub
```

Changing the URL on a Page

Sometimes you want to establish a certain look based on the page. For example, based on the type of customer signed in. You can do this programmatically in `Page_Load()`.

Use the method `PeterBlum.DES.StyleSheetManager.OverrideDefaultUrl()` to specify a different URL.

WARNING: Using `OverrideDefaultUrl()` will disable the output of compressed style sheet files into a single `<link>` tag for this page request. Instead, there will be individual `<link>` tags to the actual file contents.

[C#]

```
void OverrideDefaultUrl(PeterBlum.DES.StyleSheetFiles pStyleSheetFile, string pUrl)
```

[VB]

```
Sub OverrideDefaultUrl(ByVal pStyleSheetFile As PeterBlum.DES.StyleSheetFiles, _
    ByVal pUrl As String)
```

Parameters

pStyleSheetFile

Specifies the style sheet file's usage (association with controls). The enumerated type `PeterBlum.DES.StyleSheetFiles` has these values:

- o Calendar
- o DateAndTime
- o InteractivePages
- o Menu
- o MonthYearPicker
- o MultiSelectionCalendar
- o PopupErrorFormatter
- o PopupHints
- o SpecialDates
- o TextBoxes
- o TextCounter
- o TimePicker
- o Validation
- o VAMCompatible

pUrl

The URL used by the browser to access the file. It can use the `~` character for the web application root folder.

Example

[C#]

```
protected void Page_Load(Object sender, EventArgs e)
{
    PeterBlum.DES.StyleSheetManager.OverrideDefaultUrl(
        PeterBlum.DES.StyleSheetFiles.Calendar, "~/StyleSheets/Calendar.css");
}
```

[VB]

```
Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    PeterBlum.DES.StyleSheetManager.OverrideDefaultUrl( _
        PeterBlum.DES.StyleSheetFiles.Calendar, "~/StyleSheets/Calendar.css")
End Sub
```

Disabling the Output of Link Tags Globally

If you add the DES style sheets to other style sheet files, you may need to disable the generation of <link> tags. This can be done globally and on the page level.

Use the **web.config** file or `Application_Start()` method to disable one or more style sheet files. These settings will affect all pages in your web application, except when overridden on the page.

Using the web.config file

Use these keys in the <appSettings> section of **web.config**. Only use keys for the controls you have alternative files. Any that continue to use the default can be omitted.

```
<add key="DES_StyleSheetCalendar" value="" />
<add key="DES_StyleSheetDateAndTimeTextBoxes" value="" />
<add key="DES_StyleSheetMonthYearPicker" value="" />
<add key="DES_StyleSheetMultiSelectionCalendar" value="" />
<add key="DES_StyleSheetSpecialDates" value="" />
<add key="DES_StyleSheetTimePicker" value="" />
<add key="DES_StyleSheetInteractivePages" value="" />
<add key="DES_StyleSheetMenu" value="" />
<add key="DES_StyleSheetPopupHints" value="" />
<add key="DES_StyleSheetTextCounter" value="" />
<add key="DES_StyleSheetTextBoxes" value="" />
<add key="DES_StyleSheetPopupErrorFormatter" value="" />
<add key="DES_StyleSheetValidation" value="" />
```

Programmatically within Application_Start()

Use the method `PeterBlum.DES.StyleSheetManager.RemoveDefaultUrl()` to specify a URL of "".

[C#]

```
void RemoveDefaultUrl(PeterBlum.DES.StyleSheetFiles pStyleSheetFile)
```

[VB]

```
Sub RemoveDefaultUrl(ByVal pStyleSheetFile As PeterBlum.DES.StyleSheetFiles)
```

Parameters

pStyleSheetFile

Specifies the style sheet file's usage (association with controls). The enumerated type `PeterBlum.DES.StyleSheetFiles` has these values:

- o Calendar
- o DateAndTime
- o InteractivePages
- o Menu
- o MonthYearPicker
- o MultiSelectionCalendar
- o PopupErrorFormatter
- o PopupHints
- o SpecialDates
- o TextBoxes
- o TextCounter
- o TimePicker
- o Validation
- o VAMCompatible

Example

[C#]

```
protected void Application_Start(Object sender, EventArgs e)
{
    PeterBlum.DES.StyleSheetManager.RemoveDefaultUrl(
        PeterBlum.DES.StyleSheetFiles.Calendar);
}
```

[VB]

```
Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
    PeterBlum.DES.StyleSheetManager.RemoveDefaultUrl( _
        PeterBlum.DES.StyleSheetFiles.Calendar)
End Sub
```

Disabling the Output of Link Tags on a Page

Use the method `PeterBlum.DES.StyleSheetManager.Remove()` to stop a single file from being output on this page request.

If you want to disable all files at once, use this line:

```
PeterBlum.DES.Globals.Page.StyleSheetManager.Enabled = false
```

[C#]

```
void Remove(PeterBlum.DES.StyleSheetFiles pStyleSheetFile)
```

[VB]

```
Sub Remove(ByVal pStyleSheetFile As PeterBlum.DES.StyleSheetFiles)
```

Parameters

pStyleSheetFile

Specifies the style sheet file's usage (association with controls). The enumerated type `PeterBlum.DES.StyleSheetFiles` has these values:

- o Calendar
- o DateAndTime
- o InteractivePages
- o Menu
- o MonthYearPicker
- o MultiSelectionCalendar
- o PopupErrorFormatter
- o PopupHints
- o SpecialDates
- o TextBoxes
- o TextCounter
- o TimePicker
- o Validation
- o VAMCompatible

Example

[C#]

```
protected void Page_Load(Object sender, EventArgs e)
{
    PeterBlum.DES.StyleSheetManager.Remove(
        PeterBlum.DES.StyleSheetFiles.Calendar);
}
```

[VB]

```
Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
    PeterBlum.DES.StyleSheetManager.Remove( _
        PeterBlum.DES.StyleSheetFiles.Calendar)
End Sub
```

Browser Sensitive Style Sheet Class Names

When you assign a style sheet class name to any of Peter's Data Entry Suite's properties, you should expect that same name to be outputted to the HTML. Not every style sheet class that you define works for every browser. For example, on Netscape 7.0x browsers, there was a bug that showed border lines on tables even when the table was invisible. The best solution for that bug was to provide alternative style sheet classes for Netscape 7 that omitted the border lines.

With the Browser Sensitive Style Sheet Class Names feature, DES will switch the class name associated with a property depending on the browser. This feature is very powerful, giving you the ability to map any style sheet class name to another name. For example, DES automatically switches "DES_CalWeekRowsTable" to "DES_CalWeekRowsTable_NS70" on Calendar when it detects Netscape 7.0x.

Replacing Class Names

Any time a style sheet class name property starts with an exclamation point ("!") character, it will be reviewed for possible replacement.

Here's how use this feature.

First time – Add the CheckCssClass event

You will add an event handler to your web application that is passed the style sheet class names of any property that starts with an exclamation point. At this point, the event handler method only returns null/Nothing. In the next topic, you will add code to act on the style sheet class name.

1. Define a global method (such as in the **Global.asax** file or identified as `static/Shared` in another class) that has this format:

[C#]

```
public static string MyCheckCssClass(string pCssClass,
    System.Web.UI.WebControls.WebControl pControl,
    PeterBlum.DES.TrueBrowser pBrowser)
{
    // your evaluation of the parameters goes here.
    // return a new style sheet class name or null to keep the same name
    return null;
}
```

[VB]

```
Public Shared Function MyCheckCssClass(ByVal pCssClass As String, _
    ByVal pControl As System.Web.UI.WebControls.WebControl, _
    ByVal pBrowser As PeterBlum.DES.TrueBrowser) As String _
    ' your evaluation of the parameters goes here.
    ' return a new style sheet class name or null to keep the same name
    Return Nothing
End Function
```

2. Attach the **CheckCssClass** event handler to your method. The event is defined on the `PeterBlum.DES.StyleSheetManager` object. Do this in the `Application_Start()` method of **Global.asax**.

[C#]

```
protected void Application_Start(Object sender, EventArgs e)
{
    PeterBlum.DES.StyleSheetManager.CheckCssClass
        += PeterBlum.DES.CheckCssClassHandler(MyCheckCssClass);
}
```

[VB]

```
Sub Application_Start(ByVal sender As Object, ByVal e As EventArgs)
    AddHandler PeterBlum.DES.StyleSheetManager.CheckCssClass _
        AddressOf MyCheckCssClass
End Sub
```

Each time – Adding a new style sheet class

1. Identify any style sheet classes that do not have the right appearance in a specific browser.
2. Create a new style sheet class that offers the correct appearance. It can be in the style sheet file that has the original or in another file or <style> tag, so long as it is on the page when the control is on the page.
3. Edit the property of the control that refers to the original class name so that its name begins with an exclamation point (“!”). For example, “DESCalControl” becomes “!DESCalControl”.
4. Add logic into your MyCheckCssClass() method that evaluates the parameters and determines if it should change the class name. If it does change the name, return the new name. If it leaves it unchanged, return null/Nothing.

Here are the parameters of the MyCheckCssClass() method:

pControl

The DES control that contains the class name being evaluated. It can help optimize your function by only evaluating names based on the type of control. You will have to test the type of this control. See the example shown below.

pBrowser

The TrueBrowser object. (See “The TrueBrowser Class”.) Its properties will identify the browser type and version. You can ignore this parameter if you want to change the class based on some condition other than the type of browser.

pCssClass

The initial class name, but without the leading exclamation point. When comparing strings, be aware that users may not always follow the same case as the defined style class name. So perform case insensitive comparisons. In the example, this is done by setting *pCssClass* to lowercase and matching to lowercase versions of the name.

Example

This logic demonstrates how DES handles the class name on Calendar that is affected by Netscape 7.0. For each new name, it simply adds “_NS70” to the original name.

[C#]

```
public static string MyCheckCssClass(string pCssClass,
    System.Web.UI.WebControls.WebControl pControl,
    PeterBlum.DES.TrueBrowser pBrowser)
{
    // your evaluation of the parameters goes here.
    // return a new style sheet class name or null to keep the same name

    string vCssClassLC = pCssClass.ToLower(); // for matching case insensitively
    // Only Netscape 7.0x needs conversion because it has a bug
    // that exposes borders when hidden. So only take action
    // when it is a popup control.
    // All classes for Netscape 7.0x are pCssClass + "_NS70"
    if (vBrowser.NetscapeMoz &&
        (vBrowser.Version == new Version(7,0)))
    {
        // These are all for Calendar and only when it's a popup.
        // Skip if pControl is not that class
        if ((pControl is BaseCalendar) &&
```

```

    (((WebControl)pControl).Parent is PeterBlum.DES.BasePopupToggle))
        switch (vCssClassLC)
        {
            case "des_calweekrowstable":
                return pCssClass + "_NS70";
        }
    }
    return null;
}
}
[VB]
Public Shared Function MyCheckCssClass(ByVal pCssClass As String, _
    ByVal pControl As System.Web.UI.WebControls.WebControl, _
    ByVal pBrowser As PeterBlum.DES.TrueBrowser _
) As String
    ' your evaluation of the parameters goes here.
    ' return a new style sheet class name or null to keep the same name

    Dim vCssClassLC As String
        = pCssClass.ToLower() ' for matching case insensitively

    ' Only Netscape 7.0x needs conversion because it has a bug
    ' that exposes borders when hidden. So only take action
    ' when it is a popup control.
    ' All classes for Netscape 7.0x are pCssClass + "_NS70"
    If vBrowser.NetscapeMoz And _
        (vBrowser.Version = New Version(7,0)) Then
        ' These are all for CS_Calendar and only when it's a popup.
        ' Skip if pControl is not that class
        If (TypeOf pControl Is BaseCalendar) And _
            (TypeOf pControl.Parent Is PeterBlum.DES.BasePopupToggle)
            Select vCssClassLC
                Case "des_calweekrowstable"
                    Return pCssClass + "_NS70"
            End Select
        End If
        Return Nothing
    End Function

```

Compressing and Merging Files

Whenever DES adds its `<link>` tags to the page, it first optimizes your files for the fastest transmission with these features:

- All files are merged together, giving a single `<link>` tag on the page. That gives a single request and response with the server. If you view the HTML output of a page, look for the `<link>` tag that uses the **DES\GetFiles.aspx** or **DESGetFiles.aspx** file. *Note: DES\GetFiles.aspx uses an actual file while DESGetFiles.aspx uses an HttpHandler.*
- Files are cached in memory to limit disk based overhead.
- All comments are removed from cached files to greatly reduce their size. DES takes advantage of this feature by adding extensive comments into the files to guide you while you edit them.
- You can further reduce their size by having DES remove most of the whitespace (spaces, tabs, and carriage returns).
- For browsers that support gzip and deflate encoding (which includes modern IE, FireFox, Opera and Safari), the file will be transferred using one of those compression methods automatically. *Note: Only when using the DESGetFiles.aspx HttpHandler.*

There are times you may prefer to see individual `<link>` tags and the original (uncompressed) contents of style sheets, especially when tracking down bugs. As a result, you can turn off this feature.

Modifying the merging and compression features

Using the web.config file

Within the `<appSettings>` section of the **web.config** file, you can add this key:

```
<add key="DES_StyleSheetCompression" value="see below" />
```

Use these values in the **value** attribute:

- "SeparateFiles" or "S" – Do not merge the files. Output the original files in separate `<link>` tags. The **GetFiles.aspx** URL will not be added to the page. URLs to the actual files are returned.
- "None" or "N" – Output the merged files but do not compress them. They are unchanged aside from being merged.
- "Comments" or "C" – Merge the files, removing all comments. This is the default when the key is omitted from `<appSettings>`.
- "Full" or "F" – Merge the files, removing all comments, most space characters, tabs, and carriage returns.

Programmatically changing the settings globally and in individual web forms

If you want to set these values programmatically to affect the entire web application, you will add this line to the `Application_Start()` method of your **Global.asax** file:

```
PeterBlum.DES.StyleSheetManager.DefaultCompression =  
    PeterBlum.DES.StyleSheetCompression.value
```

where value is `SeparateFiles`, `None`, `Comments`, or `Full`.

If you want to change the compression on a single page, add this line to the `Page_Load()` method of your web form:

```
PeterBlum.DES.Globals.Page.StyleSheetManager.Compression =  
    PeterBlum.DES.StyleSheetCompression.value
```

where value is `SeparateFiles`, `None`, `Comments`, or `Full`.

Troubleshooting: How to see what DES output

When using `http://localhost`, add `desdebug=style` to the querystring of your URL. For any domain, see go to “Exploring The Current Settings: DES Debugging Reports” to setup DES’s debugging menu. Use the **style sheet** option from the DES Debug menu.

Troubleshooting: Changing the URL to GetFiles.aspx

For ASP.NET 1.x users

Sometimes the default URL to the **GetFiles.aspx** file does not work. To limit problems, you can relocate **GetFiles.aspx** into another folder, such as the web application root folder.

Then they tell DES where it is using the `<appSettings>` key in **web.config**:

```
<add key="DES_GetFilesVirtualPath" value="~/[URL]/GetFiles.aspx" />
```

Troubleshooting: Turning of Gzip/Deflate Compression

When using the `DESGetFiles.aspx` `HttpHandler` (the default for ASP.NET 2 and higher users), browsers will receive compressed data using either `gzip` or `deflate` compression. If this is causing problems, you can disable this feature.

Add this key to the `<appSettings>` section of **web.config**:

```
<add key="DES_NoResponseCompression" value="" />
```

Special Parsing Features

A style sheet file is normally static. Putting it through DES's compression system (see "Compressing and Merging Files") allows DES to parse and modify it. DES uses this to enhance the style sheet "url ()" filepath token that is often used by the background-image style.

The url () token only supports absolute and relative paths. It does not support the ASP.NET token "~/ " that converts to the web application root folder. Nor does it support the token "{APPEARANCE}" that DES maps to the Appearance folder ("~/DES/Appearance"). This parser now supports both.

- Absolute path from the web application folder can use "~/ " in place of the root folder name. When the domain root is the same as the web application root, this is replaced by "/ ". When they differ, the "~" is replaced by the folder path between domain root and web app root.

```
background-image: url ('~/DynamicData/Contents/Images/Back.gif');
```

is modified to

```
background-image: url ('/DynamicData/Contents/Images/Back.gif');
```

or

```
background-image: url ('/RootFolder/DynamicData/Contents/Images/Back.gif');
```

- When the image is located in a subfolder of the DES Appearance folder, start the URL with "{APPEARANCE}". Do not separate that token from the rest with a slash. It is inserted for you.

```
background-image: url ('{APPEARANCE}Shared/BlueDot.gif');
```

is modified to

```
background-image: url ('/DES/Appearance/Shared/BlueDot.gif');
```

or

```
background-image: url ('/RootFolder/DES/Appearance/Shared/BlueDot.gif');
```

Support for Your Own Style Sheet Files

You can take advantage of DES's compression and file merging capabilities with your own style sheets. (See "Compressing and Merging Files".) This has several benefits:

- A single <link> tag delivers all style sheets, reducing the number of http requests to the server
- All comments and most whitespace are removed from your file, allowing you to include extensive inline documentation without worrying about its transfer to the browser
- GZIP or DEFLATE compression is applied to the transfer automatically, usually reducing the transfer size by 60% or more.
- You will be adding your files programmatically. You can write logic to includes the files only when needed.

DES supports up to custom 10 style sheet file paths. They are associated with the enumerated type `PeterBlum.DES.StyleSheetFiles`, which has items for its own files (like `Calendar` and `DateTextBox`) and your files (`User1`, `User2`, ... `User10`.) You will map these `User#` types to specific URLs. For example:

`PeterBlum.DES.StyleSheetFiles.User1` maps to "`~/MyStyleSheets/StyleSheet1.css`".

Registering your Style Sheet Files

Before they can be used, you must register any style sheet files with DES. It can be done within the `web.config` file or the `Application_Start()` method of `Global.asax`.

Using web.config

In the <appSettings> section of `web.config`, add this node for each file:

```
<add key="DES_StyleSheetUser[number]" value="url" />
```

Using the example above for `User1`:

```
<add key="DES_StyleSheetUser1" value="~/MyStyleSheets/StyleSheet1.css" />
```

Using the Application_Start() method

Add code in this format:

```
PeterBlum.DES.StyleSheetManager.SetDefaultUrl(
    PeterBlum.DES.StyleSheetFiles.User[number], "url")
```

Using the example above for `User1`:

```
PeterBlum.DES.StyleSheetManager.SetDefaultUrl(
    PeterBlum.DES.StyleSheetFiles.User1, "~/MyStyleSheets/StyleSheet1.css")
```

Including your Style Sheet Files on the Page

You will be adding your files programmatically. You can write logic to includes the files only when needed. The logic can go in many places:

- The `Page_Init()` or `Page_Load()` method of the page, master page, or User Control.
- When a `DataBoundControl` (like `GridView` or `DetailsView`) adds elements that need the file, the code can go in `RowCreated` or `ItemCreated` methods.
- Within the code of a custom control that has its own styles.

Simply call this method with the `PeterBlum.DES.StyleSheetFiles` enumerated value that you registered in `web.config` or `Application_Start()`:

```
PeterBlum.DES.Globals.Page.StyleSheetManager.Include(
    PeterBlum.DES.StyleSheetFiles.User[number])
```

Using the example above for `User1`:

```
PeterBlum.DES.Globals.Page.StyleSheetManager.Include(PeterBlum.DES.StyleSheetFiles.User1)
```

Using Server.Transfer

ASP.NET 2.0 Note: This section is not needed for users of the ASP.NET 2.0 enhanced version of DES. However, it is harmless to call these methods.

On pages that use the `Server.Transfer()` method to switch to another page, additional code is required. The `Transfer()` method does not clear out the original page from memory and in fact passes the `HttpContext` with its existing **Request**, **Items** and **Handler** properties. Thus the destination page acts like its still in the original page. DES needs to reset the object it keeps in **PeterBlum.DES.Globals.Page**. You must provide code in two places.

Note: If you omit this code and the destination page uses DES's controls, JavaScript will be rendered incorrectly to the page, which means DES has no client-side presence or you will get JavaScript errors.

Prior to the Server.Transfer Call

Add this code immediately before your call to `Server.Transfer()`:

```
PeterBlum.DES.Globals.BeforeServerTransfer()
```

In the Destination Page of the Server.Transfer Call

Add this code to the beginning of your `Page_Load()` method of the destination page. (It will not be harmful if it is called when the same page is not called from `Server.Transfer()`.)

```
PeterBlum.DES.Globals.AfterServerTransfer(Page)
```

where *Page* is the current page instance. Usually this parameter is `Me.Page` in VB or `this.Page` in C#.

Using Alternative HttpHandlers (including SharePoint)

DES needs to identify the current Page object, to which it keeps a reference in **PeterBlum.DES.Globals.Page.OwnerPage**.

Normally, when a Page object is constructed, this object is the current "HttpHandler". DES retrieves the value from `HttpContext.Current.Handler` and is all set. The Page is the current HttpHandler when the browser requests a URL that specifies a web form with an **aspx** extension.

There are other types of HttpHandlers. You can create them to handle other page extensions. Third party products, like Microsoft SharePoint, also use them.

When using SharePoint or any other HttpHandler, add this line of code as the first line of your `Page_Load` method:

```
PeterBlum.DES.Globals.UsingAltHttpHandler(Page)
```

where *Page* is the current page instance. Usually this parameter is `Me.Page` in VB or `this.Page` in C#.

Note: If you fail to do this, DES will throw an exception to remind you.

Using a Redistribution License

If you have a **Redistribution License**, you must programmatically assign the serial number to the appropriate property listed below in the `Page_Load()` method. *This is required on each page that uses DES controls. It defines the pages that are licensed so that customers who receive your product and add their own pages cannot use your license on those pages.*

Product or Module	Property
Suite (complete product)	PeterBlum.DES.Globals.Page.Suite_LicenseKey
Peter's Professional Validation	PeterBlum.DES.Globals.Page.ProfessionalValidation_LicenseKey
Peter's More Validators	PeterBlum.DES.Globals.Page.MoreValidators_LicenseKey
Peter's TextBoxes	PeterBlum.DES.Globals.Page.TextBoxes_LicenseKey
Peter's Date and Time	PeterBlum.DES.Globals.Page.DateAndTime_LicenseKey
Peter's Interactive Pages	PeterBlum.DES.Globals.Page.InteractivePages_LicenseKey
Peter's Input Security	PeterBlum.DES.Globals.Page.InputSecurity_LicenseKey

Example

Suppose that you have licenses for Peter's Professional Validation with serial number 01-1212121212 and Peter's More Validators with serial number 12-2323232323.

[C#]

```
private void Page_Load(object sender, System.EventArgs e)
{
    PeterBlum.DES.Globals.Page.ProfessionalValidation_LicenseKey = "01-1212121212";
    PeterBlum.DES.Globals.Page.MoreValidators_LicenseKey = "12-2323232323";
}
```

[VB]

```
Private Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs) _
    Handles MyBase.Load
    PeterBlum.DES.Globals.Page.ProfessionalValidation_LicenseKey = "01-1212121212"
    PeterBlum.DES.Globals.Page.MoreValidators_LicenseKey = "12-2323232323"
End Sub
```

How ASP.NET Influences Peter's Data Entry Suite

Themes and Skins

ASP.NET 2.0 introduces the concepts of Themes and Skins. Microsoft built this feature to update most properties of web controls so you can build a kind of template and assign its name to the **SkinID** property. DES's controls support Themes and Skins with some limitations. Skins cannot be applied to properties in nested objects. This limitation applies to these properties:

- All Enabler, Condition, Conditions, and ExtraControlsToRunThisAction properties throughout DES controls
- Validator.ErrorMessage – See the **ErrorFormatterSkinID** property for an alternative. You can create an ErrorFormatterSkin. Then set up a skin that sets the **ErrorFormatterSkinID** property. This will load the ErrorFormatterSkin using the Themes and Skins system.
- Validator.HiliteFields
- CalculationController.Expression
- MultiFieldStateController and MultiFSCOnCommand.ControlsToChange

Automatic linking to the DES Style Sheet file

Each DES control uses style sheet definitions from several style sheet files. Normally, you have to assign a `<link>` tag to each file to use the style sheets. ASP.NET 2.0 lets the `<link>` tag be created programmatically. DES takes advantage of this capability.

See “Using Style Sheets”.

Localization

ASP.NET 2.0 introduces a syntax where the web control definitions can be mapped to a resource file (resx) that is local to the web form. The syntax looks like this:

```
<tag:controlclass runat=server meta:resourcekey="key" [other properties]>
```

You can use the Properties Editor to set the key to your resource file using the “(Expressions)” property and selecting Resources from the dropdownlist on the right. See the Visual Studio 2005 documentation for details on this syntax and how to edit it in the Properties Editor.

DES already supports localization by using its String Lookup System. (See “The String Lookup System”.) The String Lookup System has been built around the resource files stored in the **App_GlobalResources** folder. If you choose to use local resources (stored in **App_LocalResources**), you will use the features described here.

DES makes every string property that has an associated LookupID property available to the ASP.NET 2.0 localization system. For example, **Validator.ErrorMessage** has **ErrorMessageLookupID**. So ErrorMessage is supported.

Validation on AutoPostBack of the TextBox and other data entry controls

Most data entry controls that offer AutoPostBack provide these properties: **ValidationGroup** and **CausesValidation**. They set up client-side validation prior to postback. If validation fails, the post back does not occur. See “Using Validation with AutoPostBack” in the **Validation User's Guide**.

ValidationGroup property on submit controls

ASP.NET 2.0 introduces Validation Groups to its own validators. In doing so, it adds the **ValidationGroup** property to each submit control (Button, ImageButton, LinkButton). DES's submit controls already have their own property for Validation groups, **Group**. So the DES buttons now have two very similar properties.

DES will use the **ValidationGroup** property only when the **Group** property is blank. This way, you can use either. The Properties Editor will hide the ValidationGroup property to limit its use. However, Intellisense will still show both.

Page.SetFocus vs. PeterBlum.DES.Globals.Page.InitialFocusControl

The **PeterBlum.DES.Globals.Page.InitialFocusControl** property serves the same purpose as the `Page.SetFocus()` method. If you use both, they will each set up their JavaScript and the last one to run its JavaScript will establish focus.

Recommendation: Use the **InitialFocusControl** property.

XHTML Compatibility

DES generates XHTML Strict compatible code.

Obsolete features found in the ASP.NET 1.x assemblies of DES

You no longer need to be concerned with the following due to improvements Microsoft made to ASP.NET 2.0:

- The ASP.NET Design Mode Extender (“ADME”) is not used.
- When using `Server.Transfer()`, you do not need to use the methods described in “Using Server.Transfer”. You can continue to use these functions for existing code but they really do nothing.
- If you had problems where scripts were not output unless you called `PeterBlum.DES.Globals.Page.PagePreRegister()`, those pages will now work without the call to `PagePreRegister()`. Existing pages that use this function will continue to work.

Browser Support

DES scales down each of its controls depending on various characteristics of the browser. Usually this means client-side scripting is disabled. It introduces the TrueBrowser class to get an accurate description of the browser requesting the page.

Click on any of these topics to jump to them:

- ◆ The TrueBrowser Class
- ◆ Customizing A TrueBrowser Object
- ◆ Extending Support For More Browsers
- ◆ Adjusting the ErrorFormatter Based On The Browser

The following browsers support all or almost all features of DES:

- Internet Explorer for Windows, v4 and higher
- Internet Explorer for Macintosh, v5 and higher. *Limitations:*
 - No Spinners.
 - No ContextMenus
 - Keyboard commands in textboxes limited to letters and digits
 - No keyboard support in Calendar or MonthYearPicker.
- All Mozilla-based browsers including FireFox for Windows and Macintosh, Netscape for Windows and Macintosh v6 and higher, Camino (for Macintosh). *Limitations:*
 - No Spinners in Netscape 6.x or Mozilla 1.0.
 - No keyboard support in Calendar or MonthYearPicker.
- Opera for Windows, v7 and higher. *Limitations:*
 - No ContextMenus
 - Keyboard commands in textboxes limited to letters and digits
 - Keyboard support in Calendar or MonthYearPicker lacks support of arrow keys.
- Safari for Macintosh and Windows, OmniWeb 5.x (for Macintosh). *Limitations:*
 - No ContextMenus
 - No command key shortcuts
 - No keyboard support in Calendar or MonthYearPicker.
 - Safari v1.0 and 1.1 have a bug that prevents it from filtering out keystrokes in TextBoxes.

The following browsers have extensive limitations and scale down:

- Opera 5 and 6 for Windows and Macintosh. Has client-side validation support. Since these browsers do not support updating text on the client side, error messages will not support runtime tokens. No keyboard filtering on textboxes. No popups. No spinners. Many Interactive Pages features are disabled.
- Netscape 4 for Windows and Macintosh. These browsers have no client-side support. Validation is only handled on the server-side via a postback. No keyboard filtering on textboxes. No popups. No spinners. Many Interactive Pages features are disabled.
- Konqueror 2.x has many of the features, but have not been tested by PeterBlum.com. It was set up based on documentation that indicates what HTML, JavaScript and DOM features they offer.
- All other browsers are scaled down to server-side only.

The TrueBrowser Class

ASP.NET provides the browser requesting the page in the Request object's **Browser** property. However, if you have set the **ClientTarget** property on the Page object, you cannot determine the true browser because **Request.Browser** shows the browser defined in **ClientTarget**. So DES keeps browser information in the `PeterBlum.DES.TrueBrowser` class.

The `PeterBlum.DES.TrueBrowser` class has numerous properties that are grouped into three distinct types:

- Browser type and version – These are read only and reflect the true browser.
- Browser capability flags – These tell DES if the browser supports a particular feature in HTML, JavaScript or DOM.
- DES product support as a result of the Browser capabilities – These let you know which features are disabled based on the requesting browser.

You can use the **PeterBlum.DES.Globals.Page.Browser** property on to access the current TrueBrowser object of the requesting browser.

Note: A separate instance of TrueBrowser is created for each unique browser (via the HTTP_USER_AGENT string). Once created, it's cached and used by all controls, pages, and users with the same browser until it gets cleared from the cache. If you make a change to the object, the effects will impact all controls, pages, and users of the same browser.

Here are the properties on the TrueBrowser class.

Click on any of these topics to jump to them:

- ◆ [Browser Type and Version](#)
- ◆ [Browser Capabilities](#)
- ◆ [Product Feature Support](#)

Browser Type and Version

This first group tells you what browser was detected. All of these properties are read only.

Note: This URL is helpful in learning about some of the less known browsers: <http://www.quirksmode.org/>

- **UserAgent** (string) – The HTTP_USER_AGENT string passed from the browser to tell what it is. It has been set to lowercase. Use it to test a browser not listed below.
- **IE** (Boolean) – When true, it is Internet Explorer (all versions and all platforms)
- **IEWin** (Boolean) – When true, it is Internet Explorer for Windows (all versions)
- **IEWin5** (Boolean) – When true, it is Internet Explorer for Windows 5.0 and higher
- **IEWin6** (Boolean) – When true, it is Internet Explorer for Windows 6.0 and higher
- **IEWin7** (Boolean) – When true, it is Internet Explorer for Windows 7.0 and higher
- **IEMac** (Boolean) – When true, it is Internet Explorer for Macintosh (all versions)
- **NetscapeNav** (Boolean) – When true, it is Netscape Navigator v1-4
- **NetscapeMoz** (Boolean) – When true, it is Netscape version 6 and higher. The **Gecko** property will also be true.
- **Gecko** (Boolean) – When true, it is running the Gecko engine from Netscape's Mozilla project. This engine is found in FireFox, Netscape 6 and higher, the Mozilla browsers from www.Mozilla.org and other parties who license it, and Camino. If you get a Mozilla browser, this will be the only flag set to true.
- **FireFox** (Boolean) – When true, it is FireFox (all versions). The **Gecko** property will also be true.
- **FireFox15** (Boolean) – When true, it is FireFox 1.5 and higher. The **Gecko** property will also be true.
- **FireFox2** (Boolean) – When true, it is FireFox 2 and higher. The **Gecko** property will also be true.
- **FireFox3** (Boolean) – When true, it is FireFox 3 and higher. The **Gecko** property will also be true.
- **FireFox4** (Boolean) – When true, it is FireFox 4 and higher. The **Gecko** property will also be true.
- **Opera** (Boolean) – When true, it is Opera (versions up to 6)
- **OperaPresto** (Boolean) – Presto is the code name for the engine Opera introduced into version 7. When true, Opera 7 or higher is installed.
- **Opera8** (Boolean) – When true, it is Opera 8 and higher
- **Opera9** (Boolean) – When true, it is Opera 9 and higher
- **Konqueror** (Boolean) – When true, it is Konqueror (all versions)
- **AppleWebKit** (Boolean) – When true, it uses the applewebkit engine. This includes Safari, Chrome, and OmniWeb v5+. The properties below may also be set to true.
- **Safari** (Boolean) – When true, it is Apple Safari, Chrome, or OmniWeb v5.x. See also the Chrome property. When Chrome is false, its actually Apple Safari. When its true, its actually Chrome. Both browsers share the Safari drawing engine.
- **Safari2** (Boolean) – When true, it is Apple Safari 2 and higher.
- **Safari3** (Boolean) – When true, it is Apple Safari 3 and higher.
- **Chrome** (Boolean) – When true, it is Google Chrome. Note that **Safari** and **Safari2** will both be true as well.
- **Chrome2** (Boolean) – When true, it is Google Chrome v2. Note that **Safari** and **Safari2** will both be true as well.
- **OmniWeb** (Boolean) – When true, it is OmniWeb versions prior to 5. Version 5.x uses the Safari core code and is covered by the **Safari** property.

- **WebTV** (Boolean) – When `true`, it is WebTV (all versions)
- **ICab** (Boolean) – When `true`, it is ICab (all versions)
- **ICEBrowser** (Boolean) – When `true`, it is ICEBrowser (formerly ICE Storm) from Icesoft. (all versions).
- **UnknownBrowser** (Boolean) – When `true`, the UserAgent was not recognized. This TrueBrowser object will default to having most features disabled. You often use the **Customize** property to intercept this and configure the TrueBrowser object yourself.
- **Version** (System.Version) – Contains the version found. Not used by the Mozilla browsers from www.Mozilla.org.
- **GeckoVersionDate** (Int) – When **Gecko** is `true`, this is the version of the Gecko engine underlying the various browser's from the Mozilla project. It includes Netscape 6+ and FireFox. It is a date in the format YYYYMMDD and is represented as an integer here so that you can quickly compare another YYYYMMDD formatted integer to it. When **Gecko** is `false`, this is 0.
- **AppleWebKitVersion** (System.Version) – When **AppleWebKit** is `true`, this is the version of that engine. It is null when **AppleWebKit** is `false`.
- **SupportsJavaScript1_2** (Boolean) – When `true`, the browser supports JavaScript 1.2. This is a requirement for client-side validation.
- **Windows** (Boolean) – When `true`, the browser is running on the Windows Operating System.
- **Mac** (Boolean) – When `true`, the browser is running on the Macintosh Operating System.
- **Unix** (Boolean) – When `true`, the browser is running on a Unix Operating System.
- **Linux** (Boolean) – When `true`, the browser is running on a Linux Operating System.

Browser Capabilities

These properties describe specific browser capabilities that DES uses to determine its level of support. They initially are set based on the browser type and version. These properties can be modified. Any changes are immediately imposed on the properties in the Product Support section below.

These properties only affect how the server supplies HTML and JavaScript to the browser. Once on the browser, the JavaScript code detects any and all features of about the browser without any help from these properties. The goal is to allow the server to generate exactly the right code for the browser so the user only sees what the browser is capable of showing and keep the size of the data downloaded to a minimum.

Note: When “Gecko” is listed as a browser, it indicates all Gecko-based browsers. To date, that is Netscape 6+, FireFox, Mozilla, and Camino. When Safari is listed, it also includes OmniWeb 5.x.

- **SupportsClassName** (Boolean) – When `true`, the browser supports the `className` attribute on elements; in other words, style sheets are supported. When `false`, Validator controls may automatically switch to a non-style sheet setting. See “”. (Supported by IE Win 4+, IE Mac 5+, Gecko, Konqueror 2+, IceBrowser 5.2+, Opera 4+, OperaPresto, Netscape 4+, Safari)
- **SupportsInnerHTML** (Boolean) – When `true`, the browser supports the `innerHTML` attribute on elements. (Supported by IE Win 4+, IE Mac 5+, Gecko, Konqueror 2+, IceBrowser 5.2+, OperaPresto, Safari)
- **SupportsOverflowStyle** (Boolean) – When `true`, the browser supports overflow style sheet attribute. (Supported by IE Win 4+, Gecko, OperaPresto, Safari)
- **SupportsTooltip** (Boolean) – When `true`, the browser supports the `Title` attribute that defines tooltips. When `false`, the `TooltipImageErrorFormatter` may switch to another `ErrorFormatter`. See “”. (Supported by IE Win 4+, IE Mac 5+, Gecko, Konqueror 2+, Opera 3+, OperaPresto, Safari)
- **SupportsMultilineTooltip** (Boolean) – When `true`, tool tips permit carriage return characters. The tool tip properties have a token that gets substituted for a carriage return or space depending on this property. (Supported by IE Win 4+, IE Mac 5+)
- **DIVGoodContainer** (Boolean) – When `true`, the `<DIV>` tag reliably supports absolute positioning and widths. When `false`, DES may switch to using a `<TABLE>` for a popup. (Supported by all except Netscape 4.x)
- **DIVSupportsStyleVisibility** (Boolean) – When `true`, the browser supports the `style=visibility` attribute on a `<DIV>` tag reliably. When `false`, all popup features are disabled. (Supported by IE Win 4+, IE Mac 5+, Gecko, Opera 5+, Konqueror 2+, IceBrowser 5.2+, Safari)
- **TABLESupportsStyleWidth** (Boolean) – When `true`, the `<TABLE>` tag supports the `style=width` attribute reliably. When `false`, `<TABLE>` tags require the property `<TABLE WIDTH=value>`. (Supported by IE Win 4+, IE Mac 5+, Gecko, Opera 5+, Safari, Konqueror 2+, IceBrowser 5.2+)
- **OnKeyDownExtendedKeys** (string) – This is a space delimited list of extended keys supported by the browser’s `onkeydown` and `onkeypress` events. The keys can be any from this list (with the spelling shown here): UP DOWN LEFT RIGHT HOME END PAGEUP PAGEDOWN DELETE ENTER ESC HELP F1 F2 F3 F4 F5 F6 F7 F8 F9 F10 F11 F12. Context menus and tooltips with keystrokes use this to determine what to show. This way, if you define a group of keystrokes for a command like this “UP + .”, the command shown will be the first from that list that is either an alphanumeric key or in this property.

IE Win 5+ and Gecko support all of them. IE Mac 5+ supports the arrow keys. OperaPresto supports ENTER and ESC.
- **SupportsGetElementById** (Boolean) – When `true`, the browser supports the DOM function `getElementById()` for ID lookup. When this and **SupportsDocument_All** are both `false`, Validator, ValidatorSummary, and FieldStateController controller are server-side only. (Supported by IE Win 5+, IE Mac 5+, Gecko, Opera 5+, Konqueror, ICab, IceBrowser, Safari)
- **SupportsDocument_All** (Boolean) – When `true`, the browser supports the DHTML property `document.all` for ID lookup. When this and **SupportsGetElementById** are both `false`, Validator, ValidatorSummary, and FieldStateController controller are server-side only. (Supported by IE Win 4+, IE Mac 5+, ICab, IceBrowser, OmniWeb, OperaPresto.)

- **SupportsAttachEventOrAddEventListener** (Boolean) – When `true`, the browser supports the DHTML method `AttachEvent()` or the DOM method `AddEventListener()`. (Supported by IE Win 5+, Gecko, OperaPresto, Safari.)
- **SupportsOnContextMenuEvent** (Boolean) – When `true`, the browser supports the `oncontextmenu` event handler. When `false`, right click will not open a context menu. However, this doesn't stop left-button support such as when the user clicks on the Help, `PopupCalendar`, `PopupMonthYearPicker` or `QuickDateMenu` toggle button. (Supported by IE Win 5+ and Gecko.)
- **SupportsPNGImages** (Boolean) – When `true`, the browser supports PNG image files. When `false` and you supply a PNG image file, the feature will scale down to show a text label. (All buttons that support images also have a property for a text label.) (Supported by IE Win 5+, IE Mac 5+, Gecko, Netscape 4.7+, Opera 4+, Safari.)
- **SupportsDisabledOnImgAndSpan** (Boolean) – When `true`, the browser supports the `Disabled` attribute on `` and `` tags. Most browsers support the `Disabled` attribute on `<input>` tags. Microsoft has allowed it on others which makes it easy to disable onclick and keyboard event handlers. (Supported by IE Win 5.5+.)
- **SupportsFocusOnTable** (Boolean) – When `true`, the browser supports focus on `<TABLE>` tags. (Supported by IE Win 5+, FireFox 1.5+.)
- **NeedsPopupFix** (Boolean) – When `true`, the `<select>` tags appear to float above absolutely positioned elements. (Supported by IE Win 5.x, IE Win 6.x.)
- **ReliableScripting** (Boolean) – This is kind of an on/off switch for all client side scripting. Most browsers have sufficient functionality for running JavaScript. They support JavaScript v1.2 or higher and have some form of debugging tools to track down errors. When `false`, all controls are server side only. (Supported by IE Win 4+, IE Mac 5+, Gecko, Konqueror, OperaPresto, Safari)
- **SupportsXHTMLSyntax** (Boolean) – When `true`, the browser supports XHTML syntax allowing us to use `
` and `<script type='text/javascript' >` tags. (Supported by IE Win 5+, IE Mac 5+, Gecko, Konqueror, Safari, Opera 6+)
- **SupportsCallbacks** (Boolean) – *ASP.NET 2.0 only*. When `true`, the browser supports the ASP.NET 2.0 Client Callback system. (IE Win 5+, Gecko, Safari)
- **SupportsFilterStyles** (Boolean) – When `true`, the browser supports the “filter” style in cascading style sheets. It is used for special effects on these controls. (Supported by IE Win 5+)
- **SupportsStyleSheets** (Boolean) – When `true`, the browser supports style sheets through the `<link>` tag. (Supported by all exception Netscape 4.x.)
- **MakeTableInlineUsingDisplayStyle** (String) – A string used on the style "display" to make a `<table>` HTML element inline. Most use "inline". Some use "inline-block". If "", the display style cannot be made inline.
- **SupportsDisplayInlineBlockStyle** (Boolean) – When true, the display style supports the value "inline-block". (Supported by IE6+, FireFox3, Safari3, Chrome1, Opera7+, Konqueror 3.5)
- **SupportsDisplayInlineTableStyle** (Boolean) – When true, the display style supports the value "inline-table". (Supported by IE8, FireFox3, Safari3, Chrome1, Opera7+)
- **SupportsDisplayTableStyle** (Boolean) – When true, the display style supports the values "table", "table-row", and "table-cell". (Supported by IE8, FireFox, Safari3, Chrome1, Opera7+, Konqueror 3.5)
- **SupportsDisplayListItemStyle** (Boolean) – When true, the display style supports the value "list-item". (Supported by IE6+, FireFox, Safari3, Chrome1, Opera7+, Konqueror 3.5)
- **MakeWhiteSpaceNoWrapValue** (String) – A string used on the style "white-space" to prevent wrapping. Most use "nowrap". Some use "pre". If "", white-space style is not supported by the browser.

Product Feature Support

These properties determine what controls and features are scaled down or turned off. Changing the Browser Capabilities properties above automatically sets them. In addition, you can modify them if you have good reason (see the section below on adding support for a new browser.)

- SupportsClientSideValidators** (Boolean) – When `true`, the Validator controls and Conditions support their client-side code. When `false`, they only operate on the server side. Defaults to `true` when:


```
SupportsInnerHTML && ReliableScripting AND SupportsJavaScript1_2
AND (SupportsGetElementById OR SupportsDocument_All)
```
- SupportsClientSideValidationSummary** (Boolean) – When `true`, the ValidatorSummary control supports its client-side code. When `false`, it only operates on the server side. Defaults to `true` when:


```
SupportsInnerHTML && ReliableScripting AND DIVSupportsStyleVisibility
AND SupportsJavaScript1_2
AND (SupportsGetElementById OR SupportsDocument_All)
```
- SupportsFieldStateControllers** (Boolean) – When `true`, the FieldStateControllers are enabled. When `false`, they are disabled. These controls are only meaningful when they can operate on the client side so they are fully disabled without a client-side presence. Defaults to `true` when:


```
ReliableScripting AND SupportsJavaScript1_2
AND (SupportsGetElementById OR SupportsDocument_All)
```
- SupportsSpinners** (Boolean) – When `true`, the spinner control on the Numeric TextBoxes is supported. When `false`, the spinner does not get created. Defaults to `true` for the following browsers: IE Win 5+, Netscape 7+, Gecko 1.1+, OperaPresto, and Safari.
- SupportsPopups** (Boolean) – When `true`, all popup controls are supported: PopupCalendar, PopupMonthYearPicker and context menu. When `false`, they are disabled. Defaults to `true` when:


```
ReliableScripting AND DIVSupportsStyleVisibility AND
DIVSupportsAbsolutePositioning AND (SupportsGetElementById OR
SupportsDocument_All)
```
- SupportsKeyboardFiltering** (Boolean) – When `true`, the browser supports the onkeydown and onkeypress events found on textboxes. When `false`, no characters are filtered. The FilteredTextBox and Numeric TextBoxes use this. (Supported by most browsers.) Defaults to `true` when:


```
ReliableScripting AND SupportsJavaScript1_2
AND (SupportsGetElementById OR SupportsDocument_All)
```
- SupportsCalendar** (Boolean) – When `true`, Calendar and MultiSelectionCalendar controls work in on the client side. When `false`, they are a server side controls. Defaults to `true` when:


```
ReliableScripting AND SupportsInnerHTML AND (SupportsGetElementById OR
SupportsDocument_All)
```
- SupportsPopupCalendar** (Boolean) – When `true`, the PopupCalendar control is supported. When `false`, it is invisible. This affects the popup calendar feature of DateTextBox and AnniversaryTextBox as well. Defaults to `true` when:


```
SupportsPopup AND SupportsCalendar
```
- SupportsContextMenu** (Boolean) – When `true`, the ContextMenu control is supported. When `false`, it is disabled. Defaults to `true` when:


```
SupportsPopups
```

Note: When SupportsOnContextMenuEvent is false, right click context menus are disabled.
- SupportsMonthYearPicker** (Boolean) – When `true`, the MonthYearPicker control is supported on the client side. When `false`, it is a server side control. Defaults to `true` when:

ReliableScripting AND SupportsInnerHTML AND (SupportsGetElementById OR SupportsDocument_All)

- **SupportsTimePicker** (Boolean) – When true, the TimePicker control is supported on the client side. When false, it is a server side control. Defaults to true when:

ReliableScripting AND SupportsInnerHTML AND (SupportsGetElementById OR SupportsDocument_All)

- **SupportsClientSideCreatesHTML** (Boolean) – When true, the browser supports all of the scripting demanded to create the control in HTML using client-side code. When false, these controls require the server side code generate the HTML and scripts. Generally browsers without support for innerHTML need this set to false. Defaults to true when:

SupportsInnerHTML AND (NOT IEMac)

Customizing A TrueBrowser Object

Each time a new UserAgent is provided from a browser, DES creates a unique TrueBrowser object with default settings based on its understanding of that browser. You may want to add or remove support for features by changing the TrueBrowser object's properties. DES offers you a delegate on **PeterBlum.DES.TrueBrowser.Customize** which is called for each unique UserAgent. It is passed the TrueBrowser object. Your code reviews the current settings such as the UserAgent, browser type, and browser version to adjust the TrueBrowser object.

You set up the **Customize** property in the `Application_Start()` method of your **Global.asax** file.

The delegate for the **Customize** property is as follows:

[C#]

```
public delegate void TrueBrowserCustomizer(PeterBlum.DES.TrueBrowserArgs pArgs);
```

[VB]

```
Public Delegate Sub TrueBrowserCustomizer( _
    ByVal pArgs As PeterBlum.DES.TrueBrowserArgs)
```

The `PeterBlum.DES.TrueBrowserArgs` class is defined as follows:

[C#]

```
public class TrueBrowserArgs : EventArgs
{
    public TrueBrowser TrueBrowser { get; }
}
```

[VB]

```
Public Class TrueBrowserArgs Inherits EventArgs
    Public Property TrueBrowser As TrueBrowser
End Class
```

Example

This will detect IE Mac 5 and remove support for keyboard filtering on textboxes (not necessarily something you would do in the real world). All of this code is added to **Global.asax**.

[C#]

In `Application_Start()`:

```
PeterBlum.DES.TrueBrowser.Customize =
    new PeterBlum.DES.TrueBrowserCustomizer(BrowserDetector);
```

...

```
public void BrowserDetector(PeterBlum.DES.TrueBrowserArgs pArgs)
{
    if (pArgs.TrueBrowser.IE && pArgs.TrueBrowser.Mac &&
        (pArgs.TrueBrowser.Version.Compare(new Version(5, 0)) == 0))
    {
        pArgs.TrueBrowser.SupportsKeyboardFiltering = false;
    }
}
```

[VB]

```
In Application_Start():
```

```
PeterBlum.DES.TrueBrowser.Customize = _  
    New PeterBlum.DES.TrueBrowserCustomizer(AddressOf BrowserDetector)
```

```
...
```

```
Public Sub BrowserDetector(ByVal pArgs As PeterBlum.DES.TrueBrowserArgs)  
    If (pArgs.TrueBrowser.IE And pArgs.TrueBrowser.Mac And _  
        (pArgs.TrueBrowser.Version.Compare(New Version(5, 0)) = 0)) Then  
        pArgs.TrueBrowser.SupportsKeyboardFiltering = False  
    End If  
End Sub
```

Handling UnknownBrowsers

When `TrueBrowser.UnknownBrowser` is `true`, you may assist DES to customize the `TrueBrowser` object. You will still use the `Customize` property. The `TrueBrowser` object, passed to your `Customizer` method, offers the method `CheckBrowserType()` which assists you in querying the `UserAgent` to detect the browser you are looking for. You must know of a unique string within the `UserAgent` that is followed by the version number to use this method. PeterBlum.com often refers to http://www.zytrax.com/tech/web/browser_ids.htm as a source for this information.

Example

In this example, `UnknownBrowser` is detected. The code then looks for the `AmigaVoyager` browser whose `UserAgent` looks like this:

AmigaVoyager/3.4.4 (MorphOS/PPC native)

[C#]

In `Application_Start()`:

```
PeterBlum.DES.TrueBrowser.Customize =
    new PeterBlum.DES.TrueBrowserCustomizer(BrowserDetector);
```

...

```
public void BrowserDetector(PeterBlum.DES.TrueBrowserArgs pArgs)
{
    if (pArgs.TrueBrowser.UnknownBrowser &&
        pArgs.TrueBrowser.CheckBrowserType("AmigaVoyager"))
    { /* note: These settings are for the example only */
        pArgs.TrueBrowser.ReliableBrowser = true;
        pArgs.TrueBrowser.SupportsTooltip = true;
        pArgs.TrueBrowser.SupportsJavascript1_2 = true;
    }
}
```

[VB]

In `Application_Start()`:

```
PeterBlum.DES.TrueBrowser.Customize = _
    New PeterBlum.DES.TrueBrowserCustomizer(AddressOf BrowserDetector)
```

...

```
Public Sub BrowserDetector(ByVal pArgs As PeterBlum.DES.TrueBrowserArgs)
    If (pArgs.TrueBrowser.UnknownBrowser And _
        pArgs.TrueBrowser.CheckBrowserType("AmigaVoyager")) Then
        ' note: These settings are for the example only
        pArgs.TrueBrowser.ReliableBrowser = True
        pArgs.TrueBrowser.SupportsTooltip = True
        pArgs.TrueBrowser.SupportsJavascript1_2 = True
    End If
End Sub
```

Extending Support For More Browsers

These controls are designed to look at browser features in JavaScript, not the browsers themselves. So while Netscape 4 doesn't support the DOM attribute "innerHTML", the code doesn't check for Netscape. It checks for innerHTML support. The server side code uses the properties of TrueBrowser to determine the support and TrueBrowser was coded "at the factory" with browsers and operating systems PeterBlum.com had access to. There are many current and future browsers that may work well with DES that haven't been tested. This section will help you add support for these browsers.

On the client side, most of the JavaScript code uses object detection instead of the capability flags within TrueBrowser. For example, to determine if the DOM function `document.getElementById` exists, the code does this:

```
if (document.getElementById) { vField = document.getElementById("xyz"); }
```

To add support for a browser, follow these steps:

1. Set up a test page with the desired controls.
2. In the `Page_Load()` method, get the TrueBrowser object from the **PeterBlum.DES.Globals.Page.Browser** property.
3. Update the Browser Capabilities properties in the TrueBrowser object to reflect the browser that you are testing.
4. Test the control. Once it is validated, use the Customize property to establish your settings on all pages. See "The TrueBrowser Class".
5. Notify PeterBlum.com at Contact@PeterBlum.com with any information about success or failure of your tests. Peter may offer to work out any bugs or limitations found if you are interested in being the primary source of testing.

Adjusting the ErrorFormatter Based On The Browser

ErrorFormatters take advantage of some features unique to modern browsers, such as style sheet class names, tooltips and JavaScript. When your page appears on a browser that doesn't support the ErrorFormatter, the user may not see an error message or it will not have the correct style sheet.

- `PopupErrorFormatter` requires modern style sheet attributes. Without it, the user only sees the image.
- `AlertImageErrorFormatter` and `HyperLinkErrorFormatter` require JavaScript functionality to show the alert. Without the alert, the user only sees the image.
- `TooltipImageErrorFormatter` requires support of tooltips (the `Title` attribute) that many earlier browsers lack. Without the tooltip, the user only sees the image.
- `TextErrorFormatter` uses a style sheet class name to establish its style by default. It requires the `className` attribute that earlier browsers lack. Without the class name, the text does not stand out.

DES can automatically correct the ErrorFormatter for these browsers to assure you that all users get the desired error message. It has an event handler, **`PeterBlum.DES.Globals.OnAdjustValidatorActionToBrowser`**, which is called on each Validator to detect problems with the ErrorFormatter and update the ErrorFormatter's properties or replace the ErrorFormatter.

By default, `OnAdjustValidatorActionToBrowser` uses the method

`PeterBlum.DES.Globals.DefaultAdjustVAToBrowser()`. This method performs the following actions and offers properties to customize its behavior:

- If **`TrueBrowser.SupportsJavaScript1_2`** is `false` and the ErrorFormatter is `AlertImageErrorFormatter` or `HyperLinkErrorFormatter`, it can switch to a new ErrorFormatter. The property **`PeterBlum.DES.Globals.ErrorFormatterWhenNoJavaScript`** can contain an instance of a replacement ErrorFormatter object. If it's `null/nothing`, no replacement happens. It defaults to `null/nothing`. All compatible properties are copied from the original `AlertImageErrorFormatter` to the new ErrorFormatter. *Note: If you switch to `TextErrorFormatter`, your page will likely need more space for the error message.*
- If **`TrueBrowser.SupportsPopup`** is `false` and the ErrorFormatter is `PopupErrorFormatter`, it can switch to a new ErrorFormatter. The property **`PeterBlum.DES.Globals.ErrorFormatterWhenNoPopup`** can contain an instance of a replacement ErrorFormatter object. If it's `null/nothing`, no replacement happens. It defaults to `AlertImageErrorFormatter`. All compatible properties are copied from the original `PopupErrorFormatter` to the new ErrorFormatter. *Note: If you switch to `TextErrorFormatter`, your page will likely need more space for the error message.*
- If **`TrueBrowser.SupportsTooltip`** is `false` and the ErrorFormatter is `TooltipImageErrorFormatter`, it can switch to a new ErrorFormatter. The property **`PeterBlum.DES.Globals.ErrorFormatterWhenNoTooltip`** can contain an instance of a replacement ErrorFormatter object. If it's `null/nothing`, no replacement happens. It defaults to `AlertImageErrorFormatter`. All compatible properties are copied from the original `TooltipImageErrorFormatter` to the new ErrorFormatter. *Note: If you switch to `TextErrorFormatter`, your page will likely need more space for the error message.*
- If **`TrueBrowser.SupportsClassName`** is `false` and **`BaseErrorFormatter.CssClass`** is assigned, it can apply a new set of styles that do not use the style sheet files. The property **`PeterBlum.DES.Globals.NoStyleSheetErrorFormatterControlStyle`** is a `System.Web.UI.Style` object where you can assign properties like `ForeColor` and `Font.Name`. It defaults to a `ForeColor` of `Color.Red`.

Set the properties of **`PeterBlum.DES.Globals`** in your `Application_Start()` method of the **`Global.asax`** file. These properties are all static/shared.

You can provide a substitute event handler to **`OnAdjustValidatorActionToBrowser`**. The delegate definition is:

[C#]

```
public delegate void AdjustValidatorActionToBrowser(
    BaseValidatorAction action);
```

[VB]

```
Public Delegate Sub AdjustValidatorActionToBrowser ( _
    ByVal action As BaseValidatorAction)
```

action

BaseValidatorAction is an internal class that does most of the work of a Validator control. It contains most of the same properties, such as **ErrorFormatter**, which you will modify or replace in your event handler.

To attach your event handler method to **OnAdjustValidatorActionToBrowser**, do this in the `Application_Start()` method of the **Global.asax** file. *Note: this property does not allow multiple event handlers.*

[C#]

```
PeterBlum.DES.Globals.OnAdjustValidatorActionToBrowser =  
    new AdjustValidatorActionToBrowser(MyMethod);
```

[VB]

```
PeterBlum.DES.Globals.OnAdjustValidatorActionToBrowser = _  
    New AdjustValidatorActionToBrowser(AddressOf MyMethod)
```

Your method can call the default method, `PeterBlum.DES.Globals.DefaultAdjustVAToBrowser()`.

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. See also the “Troubleshooting” section of the **Installation Guide**, especially for error messages involving assemblies loading.

Click on any of these topics to jump to them:

- ◆ [Handling JavaScript errors](#)
- ◆ [Exploring The Current Settings](#)
- ◆ [Common Error Messages](#)
- ◆ [Runtime Problems](#)
- ◆ [Design Mode Problems](#)

Handling JavaScript errors

You can detect a JavaScript error in Internet Explorer by going to its **Tools; Internet Options** menu and selecting the **Advanced** tab. Then do the following:

- Mark “Display a notification about every script error”. This will show an alert box briefly describing the error when it occurs.
- If you use Visual Studio.net (any version), unmark “Disable Script Debugging”. This will let you launch Visual Studio.net’s powerful debugger to use with Internet Explorer. When you get an error message, click Yes then select Visual Studio. You will be placed on the line with the error and be able to use the same debugging tools you use with VB and C# to view the call stack and variables.

There are several causes to a JavaScript error or reasons why JavaScript fails to run.

1. The control is being updated by an AJAX system, and the error occurs after a callback. You have not set up these controls to correctly use the DES AJAX features. See “Using These Controls with AJAX”.
2. You have set up for AJAX, but the setup missed something. It is very easy to get JavaScript errors when these controls are not correctly set up to work within AJAX. Here are some things to look for.
 - The `PeterBlum.DES.AJAXManager.UsingXYZ()` method is not in `Page_Load()` or the `PageManager` control does not use the **AJAXManager** property correctly.
 - When using `AJAXManager.UsingAJAXUpdates()`, it may not be properly configured.
 - The `AJAXManager.UsingXYZ()` method is in `Page_Load()` but is not being called because it is nested in an IF statement.
 - The `AJAXManager.UsingXYZ()` method is called after another method on `AJAXManager`. It should be the first statement.
 - When **AJAXManager.AllInAJAXUpdate** is false, all web controls that interact with an AJAX update must have their **InAJAXUpdate** property set to true. To determine if this is the case, set **AJAXManager.AllInAJAXUpdate** to true. If the problem goes away, you know to look for a DES control that needs its **InAJAXUpdate** property set to true. Set `<%@ Page Trace=True %>` then run the page. The trace will identify each DES control using `InAJAXUpdate`.
 - A control or feature is first added to the page as the result of a callback. That type of control or feature must be identified in `Page_Load()` each time it’s called by using the `AJAXManager.PreregisterForAJAX()` method. It’s common to overlook `Hints` and `AutoPostBack` (setup on `TextBoxes`, `MultiSegmentDataEntry`, and `NativeControlExtender` controls) and `DisableOnSubmit` (setup on `Buttons`).
3. The browser does not find the JavaScript files provided by DES. This is a setup problem and will occur on all pages using DES. By default, DES maintains its scripts inside resources of the **PeterBlum.DES.dll** assembly file and uses the **GetFiles.aspx** file in ASP.NET 1.x or an `HttpHandler` in ASP.NET 2.0 to retrieve them together in a single web request.

ASP.NET 2+ Users

The **web.config** file should have this `HttpHandler` setup in the `<httpHandlers>` section:

```
<add path="DESGetFiles.aspx" verb="GET"
      type="PeterBlum.DES.GetFilesHandler, PeterBlum.DES" />
```

If its missing, either add it manually or run the **Web Application Updater** with the “Update a web application” option.

If you are using IIS 7, you also have this tag in the `<handlers>` section of `<system.webServer>` section of the **web.config** file. If it is missing, see “Using IIS 7” in the **Installation Guide**.

If this still does not work, you can switch DES to using the **GetFiles.aspx** file instead of the `HttpHandler`. Add this to the `<appSettings>` section of the **web.config** file.

```
<add key="DES_GetFilesVirtualPath" value="~/DES/GetFiles.aspx" />
```

ASP.NET 1.x Users

Start by following these steps:

- Open the page in the browser.
- Use the browser's View Source command.
- Locate the text "GetFiles.aspx" within a <script> tag. It is part of a URL. Review that URL against your web site and IIS configuration. If not found, see #5 below.
- Enter that URL within the browser. Start with your domain, followed by the URL. For example:
http://localhost/WebSite1/DES/GetFiles.aspx?various parameters
- If it comes up with a page containing this heading, then the scripts are loading.

If it reports an HTTP 404 error, the URL is not working. Consider these issues:

- IIS is mapping the URL to a different location of the **GetFiles.aspx** page. See "Troubleshooting: Changing the URL to GetFiles.aspx".
 - IIS does not allow access to the folder containing the **GetFiles.aspx** page. Correct its security or move the file using "Troubleshooting: Changing the URL to GetFiles.aspx".
 - You are using the DES_SeparateScripts key in the <appSettings> section of the **web.config** file. *This occurs when using the source code for the script files.* The key tells DES to retrieve actual script files instead of the resources. It uses another key, DES_ScriptVirtualPath, to identify the URL to the folder containing those scripts. Correct the URL.
4. The page is incorrectly structured to run JavaScript. Some users have omitted their <html> tags or positioned them WITHIN the <form> tags. Internet Explorer will not run any <script> tags that are outside of the <html> tag.
 5. It is possible that the <script> tag that loads the **GetFiles.aspx** file is not found on the page. To determine this, run the page and use View Source. Search for the text "GetFiles.aspx". If it is not found inside a <script> tag, this is the problem.

All Users

When using Forms Authentication, users have found that their Page_Load() method is invoked several times due to URLs requesting files that require authentication to have been approved. Your pages look for files in the DES folder, your own images, style sheet, and themes folders. This problem is especially noticeable on a login page because login is run when authentication has not been approved.

Solution: Make sure your DES folder and these other file support folders are given anonymous access. See "Images and style sheets do not load when the site uses Forms Authentication".

ASP.NET 1.x Assembly Users

This happens when the page is structured in an unusual way. It is usually caused by the last web control on the page having a bug that does not call its own PreRender methods. Try each of these three ways to fix this:

- The </form> tag should not be flush with the prior web control on the page. Be sure there is at least one ' ';' before and after the </form> tag.
- Add a Placeholder control immediately before the </form> tag. The Placeholder does not change your web form's output but provides a working PreRender event handler.
- On the page, override the Render() method like this:

```
[C#]
protected override void Render(HtmlTextWriter pWriter)
{
    PeterBlum.DES.Globals.Page.AddToPage();
    base.Render(pWriter);
}
```

[VB]

```
Overrides Protected Sub Render(ByVal pWriter As HtmlTextWriter)
    PeterBlum.DES.Globals.Page.AddToPage()
    MyBase.Render(pWriter)
End Sub
```

- Your site uses Forms Authentication to force a login as the user goes to a secure page. Often users instruct Forms Authentication to protect all files and folders, except those containing script, image, and style sheet files. DES has image and style sheet files too. They are in the **[web application folder]/DES** folder. Make sure that folder is not protected by Forms Authentication.

In the <configuration> section of your **web.config** file, add this to exclude the **DES** folder.

```
<location path="DES">
  <system.web>
    <authorization>
      <allow users="*" />
    </authorization>
  </system.web>
</location>

<location path="DESGetFiles.aspx">
  <system.web>
    <authorization>
      <allow users="*" />
    </authorization>
  </system.web>
</location>
```

- If the error is “Unterminated string constant”, you have embedded a carriage return character into a string property such as **ErrorMessage**, **SummaryErrorMessage**, **Label.Text**, **ErrorFormatter.BeforeHTML**, **ErrorFormatter.AfterHTML**, or **ErrorFormatter.Tooltip**. Look at your web form in HTML text view. Search for the string “” and “
”. Remove them or replace them with “
”.
- If you create a control programmatically and do not assign a value to the ID property, Microsoft’s web controls often omit adding the HTML attribute `id=` into the tag that describes that web control. If you attach that control to any of DES’s controls (such as to the `Validator.ControlToEvaluate` property), it will generate a JavaScript error because the client-side code cannot find your web control’s HTML without the `id=` attribute. Be sure to assign the ID property on each web control you associate with DES controls.
- Your page was called using `Server.Transfer()`. See “Using `Server.Transfer`”.
- If the control is in a `DataGrid` or `DataList`, determine if you have set the containing column’s **Visible** property to `false`. If it is in the header or footer row, determine if the **ShowFooter** or **ShowHeader** property is `false`. In each of these cases, the `DataGrid` and `DataList` function incorrectly by asking third party controls to write their JavaScript out even though the control will never render its HTML.

To fix this, set the **Visible** property to `false` on the DES control when the column, header or footer is invisible. Generally this should be done in the `ItemDataBound` event.

In this example, there are three `DateTextboxes`. ID=“HeaderDTB” is in the Header. ID=“FooterDTB” is in the Footer. ID=“ColumnDTB” is in the 2nd column.

[C#]

```
protected void Grid_ItemDataBound(object sender, DataListItemEventArgs e)
{
    Control vDTB = e.Item.FindControl("HeaderDTB");
    if (vDTB != null)
        vDTB.Visible = Grid1.ShowHeader;
    vDTB = e.Item.FindControl("FooterDTB");
    if (vDTB != null)
        vDTB.Visible = Grid1.ShowFooter;
    vDTB = e.Item.FindControl("ColumnDTB");
    if (vDTB != null)
        vDTB.Visible = Grid1.Columns[1].Visible;
}
```

[VB]

```
Protected Sub Grid_ItemDataBound(ByVal sender As object, _
    ByVal e As DataListItemEventArgs)
    Dim vDTB As Control = e.Item.FindControl("HeaderDTB")
    If Not vDTB Is Nothing Then
        vDTB.Visible = Grid1.ShowHeader
    End If
    vDTB = e.Item.FindControl("FooterDTB")
    If Not vDTB Is Nothing Then
        vDTB.Visible = Grid1.ShowFooter
    End If
    vDTB = e.Item.FindControl("ColumnDTB")
    If Not vDTB Is Nothing Then
        vDTB.Visible = Grid1.Columns[1].Visible
    End If
End Sub
```

11. There was other JavaScript on the page that generated an error. Once a JavaScript error occurs, all remaining code is aborted for the current action. So if there was JavaScript code that failed during initialization, DES will not get a chance to run its own initialization code. You will have to debug the JavaScript that isn't part of DES to fix the problem. If you have Visual Studio, see the instructions on the top of this topic to set up the Visual Studio debugger with Internet Explorer. If you have FireFox, get the FireBug add-in and use its debugger.
12. Look on the page for HTML comment tags: `<!-- HTML in here -->`. If they are enclosing any ASP.NET web controls (anything that uses the `<tagprefix:classname>` syntax), this is a problem. ASP.NET ignores HTML comment tags and lets web controls output their HTML and scripts to the page. The HTML is inside those comment tags. When some JavaScript attempts to access that HTML (by using `document.getElementById()` or `DES_GetById()`), it will not be found.

The solution is to switch from HTML comment tags to ASP.NET server side comment tags:

```
<%-- HTML in here --%>
```

This completely removes the ASP.NET web controls from the page output.

ORIGINAL:

```
<!--
<des:RequiredTextValidator id="Validator1" runat="server"
    ControlIDToEvaluate="TextBox1" ErrorMessage="Required" />
-->
```

CORRECTED:

```
<%--
<des:RequiredTextValidator id="Validator1" runat="server"
    ControlIDToEvaluate="TextBox1" ErrorMessage="Required" />
--%>
```

What to do when none of the suggestions above work

It is time to get PeterBlum.com tech support involved. Here's what Peter will need from you.

1. A clear description of the problem including what controls are involved and any special settings they are using (like AJAX).
2. The actual web form, including its user controls and code.
3. A URL to the actual live page, if possible OR a capture of the page by following these steps.
 - a. Open the page in Internet Explorer 6+.
 - b. Go through the steps it takes to reproduce the error. This may involve post backs. The goal is to have the browser contain the exact HTML that contains the script error.
 - c. Use the browser's **File; Save As** command with **Save As Type=Web Page, Complete**.
 - d. One HTML file and folder will be created. Zip both of them up.
 - e. Provide tech support with that Zip file.
4. Step-by-step instructions for reproducing the problem on the page. If the error occurs as the page is loaded, please indicate that.
5. Email tech support at support@PeterBlum.com.

Exploring The Current Settings: DES Debugging Reports

DES has a special feature that lets you review its various components on the server. It's called DES Debugging Reports.

Go to any page using DES controls and add a special querystring parameter to the URL. It will return a page detailing a certain aspect of DES's setup. This is very helpful after site deployment and for technical support.

The DES Debugging Reports expose the following information:

- **Globals** – Settings from the **Global Settings Editor** and more. Lists the values of the **PeterBlum.DES.Globals** object, many of which are loaded from the **custom.des.config** file and **web.config** file.
- **Page-Level** – Property values on the **PeterBlum.DES.Globals.Page** object. If you are using the PageManager control, its settings will be reflected here too.
- **Validation** – Identifies all validators, buttons, ValidationSummary and CombinedErrorMessage controls on the page. Provides key settings to help you understand how they are working. A great way to determine why a button isn't firing validators.
- **AJAX** – How DES is setup to work with AJAX on this page. It identifies the framework and several key properties. It lists preloaded features. If the **AJAXManager.AllInAJAXUpdate** property is `false`, it also lists every DES control on the page, with its **InAJAXUpdate** property. Use it to determine if a control should be changed to **InAJAXUpdate=true**. It helps debug AJAX related problems and helps you optimize the performance of the page.
- **Overall configuration** – Identifies the file paths and URLs for the configuration files, Appearance Folder, Licenses folder, and the **GetFiles.aspx** form. Identifies the DES assemblies, by version and location. Lists all keys defined in the <appSettings> section of the **web.config** file.
- **Licenses** – Identifies the license files found, the License Keys, and which licenses are in use. For users of Web Server licenses, it is particularly valuable after initial deployment to production to be sure your production license files are in use instead of the limited non-production licenses.
- **Style Sheets** – Since style sheets can be merged and compressed, it helps to know if your styles are being delivered to the page. This lists the settings used by style sheets, including the URLs and file paths used to retrieve them. It also outputs the same text from your files that is sent to the browser.

Click on any of these topics to jump to them:

- ◆ Running a DES Debugging Report from `http://localhost`
- ◆ Running a DES Debugging Report from when the Server is not local
 - Access by known IP addresses
 - Access by password

Running a DES Debugging Report from http://localhost

When the page is run from http://localhost (or the IP address requesting the page is 127.0.0.1), just add the parameter DESDebug= to the URL of the page. It will give you the DES Debug menu.

Note: This feature can be disabled by using the DES_DebugAllowedIPs key in <appSettings> without declaring the IP address 127.0.0.1.

Example

```
http://localhost/myfolder/myform.aspx?DESDebug=
```

Running a DES Debugging Report from when the Server is not local

Often in a development environment, the development web server is not accessed through `http://localhost`. In that case, you can either expose the `DESDebug=` parameter based on the IP address, or create a custom parameter name that is in effect, a password, so you can run these commands on a production server.

In addition, when the site is put in production, it is not safe to allow site visitors to use the `DESDebug=` parameter, because it exposes sensitive information, like file paths on your server. The `DESDebug=` parameter will only work if you are coming from a known IP address. For any other IP address, you create a custom parameter name.

Click on any of these topics to jump to them:

- ◆ Access by known IP addresses
- ◆ Access by password

Access by known IP addresses

You will define the `DES_DebugAllowedIPs` key in the `<appSettings>` section of the **web.config** file with a semicolon-delimited list of IP addresses. These are the only IP addresses that support the `DESDebug=` querystring parameter.

Once setup, you use this feature by added `DESDebug=` to the querystring of your page's URL. It will display the DES Debug Menu. For example:

```
http://www.mydomain.com/myfolder/myform.aspx?DESDebug=
```

Setting up IP Addresses

Add the `DES_DebugAllowedIPs` key in the `<appSettings>` section of the **web.config** file with a semicolon-delimited list of IP addresses. Explicitly include the IP address `127.0.0.1` to support `http://localhost`. Omit it if you do not want support `http://localhost`.

Example with one IP Address:

```
<add key="DES_DebugAllowedIPs" value="127.0.0.1" />
```

Multiple IP Addresses can be specified in two ways:

- Semicolon delimited list. For example:

```
<add key="DES_DebugAllowedIPs" value="127.0.0.1;127.0.0.13;155:35:10:01" />
```

- Partial IP address. Only provide the first few segments of the IP address. All IPs matching those segments are used. Always provide a trailing period. For example:

```
<add key="DES_DebugAllowedIPs" value="127.0.0." />
```

You have the option of setting this programmatically during application startup. Assign your path to this static/shared property: **PeterBlum.DES.DebugManager.AllowedIPs**. It accepts a string in the same format as described above.

```
PeterBlum.DES.DebugManager.AllowedIPs = "127.0.0.1;127.0.0.13"
```

Access by password

Security should be a concern, because the information displayed by these features exposes some information about your server (particularly filepaths and how you have setup these web forms.) As a result, the `DESDebug=` parameter can only be used from specific IP addresses, usually development computers within your organization. If you want to allow access from any other computer, you need to define a password which becomes querystring parameter that replaces “DESDebug”.

SECURITY NOTES: There is no backdoor code that allows PeterBlum.com to see this information through your site. This prevents hackers from discovering a backdoor. If you work with tech support, you may be asked to provide the information returned from these commands. Just provide the page output, not the URL with the querystring parameter.

Once setup, add your password + “=” to the querystring of the URL of a page. It will display the DES Debug menu.

Example using the password *ickypark1*:

```
http://www.mydomain.com/myfolder/myform.aspx?ickypark1=
```

Setting up the Password

1. Determine a short password that will be used as your querystring parameter. It must be only alphanumeric (compatible with HTML querystring parameter names). While a name like “debugDES” might sound like a good idea, the entire reason for having a password as the parameter name is because this feature *exposes information about your server such as file paths that you don't want to expose to hackers who use an obvious parameter name.*
2. Add this line to the `<appSettings>` section of your **web.config** file:

```
<add key="DES_DebugParameterName" value="your password" />
```

If you are testing a different server, don't forget to update that server's **web.config** file.

Alternatively, you can set the password in the **PeterBlum.DES.DebugManager.ParameterName** property in the `Application_Start()` method.

Common Error Messages

The page generates this error: "Access is denied: 'PeterBlum.xyz.dll'."

The error message is identified as a "Configuration Error" and identifies the "Source Error" as `<add assembly="*" />` in the source file **machine.config**.

This is a very common error in ASP.NET and can happen with almost any third party control that you include in your application. It's usually due to the Indexing Service or a virus scanner that is scanning the **[windows]\microsoft.net\framework\[version]\temporary asp.net files** folder although other applications can be its source.

Use this Microsoft Knowledge Base article to better understand it:

<http://support.microsoft.com/default.aspx?scid=kb;en-us;329065>

Your task is to prevent the conflicting software from accessing files in the **temporary asp.net files** folder.

The page generates this error: "Metadata file 'c:\windows\microsoft.net\framework\[version]\temporary asp.net files\[more path info]\peterblum.xyz.dll' could not be found"

This is a strange and poorly documented behavior of ASP.NET. ASP.NET copies assemblies from the **\bin** folder into the specified folder as the assembly is used by your web site. This error indicates that ASP.NET fails to actually copy the file or the file is deleted out from under ASP.NET. The cause has yet to be determined, but it can happen on any assembly found in the **\bin** folder.

Customers who encountered this error have found these solutions worked for them:

- The following should be removed from the **web.config** file:


```
<identity impersonate="true" />
```
- Install all Peter's Data Entry Suite assemblies in the Global Assembly Cache. Remove them from the **\bin** folder. Assemblies in the GAC are not copied into the **Temporary ASP.NET files** folder.

The page or design mode generates this error "File or assembly name PeterBlum.DES, or one of its dependencies, was not found."

This error may be a result of a number of things:

1. Your web application was set up to look for the **PeterBlum.DES.dll** in the **\bin** folder but it is not there. Copy it from **[DES Installation folder]\Assemblies**.
2. You have installed the **PeterBlum.DES.dll** into the Global Assembly Cache.
3. The `<% @Register tagPrefix="des" %>` tag contains a specific version in the `Assembly=` parameter that does not match the current **PeterBlum.DES.dll**'s version. Usually you can leave this unchanged.

See "What to do when you get version errors".

The page or design mode generates this error "The located assembly's manifest definition with name [assembly name] does not match the assembly reference."

This means that the assembly identified as [assembly name] was compiled to use another assembly that has a particular version. While the assembly was found, the version does not match the version that [assembly name] was compiled with.

See "What to do when you get version errors".

The page generates this runtime error: "Operation could destabilize the runtime."

This is a `System.Security.VerificationException` that happens in `Application_Start` when the first line of DES code is invoked. There are numerous ways for ASP.NET to report this. You may need to do some web searches to find the one that relates to your situation. Here are some solutions my customers have found.

- Reboot and recompile

- It is an incompatibility with a setting in IIS 7.5. The default in IIS7.5 is to run all applications under an application pool and that application pool to use a 'virtual user account'. This virtual user then has to have rights given to the folder and have the 'load profile' turned on.

Alternatively, you can click advanced settings on a website and give it an admin username/password to 'run as' and the problem also goes away.

<http://learn.iis.net/page.aspx/624/application-pool-identities/>

- In IIS, change the Application Pool to run as Network Server rather than Application Pool Identity.

The page generates this runtime error: "Type 'PeterBlum.DES.class' does not have a property named 'propertyname'"

You have entered the property name incorrectly.

The page generates this error: "Request for the permission of type 'System.Web.AspNetHostingPermission, System, Version=2.0.0.0, Culture=neutral, PublicKeyToken=b77a5c561934e089' failed. [filepath]"

Only in ASP.NET 2.0. This indicates that the filepath specified is not correctly configured to run ASP.NET applications from this computer. It usually happens when the filepath is a URL (like <http://otherserver/file>) or a shared filepath (like `\\computername\sharedname\folder`).

- Go to the Microsoft .net Framework 2.0 Configuration program
- Expand its tree to this path: `\MyComputer\Runtime Security Policy\Machine\Code Groups\All Code\LocalInternet Zone`
- Add a new node that defines the URL or UNC path to use "FullTrust" permission.

The page generates this runtime error: "DES does not know what the current Page object is. Please call PeterBlum.DES.Globals.UsingAltHttpHandler(Page) as the first line of your Page_Load method."

See "Using Alternative HttpHandlers (including SharePoint)".

Visual Studio generates this error: Warning: The dependency 'PeterBlum.DES, Version=4.0.#.5000' cannot be copied to the run directory because it would overwrite the reference "PeterBlum.DES, Version=4.0.#.5000"

This warning message is harmless. It tells you that Visual Studio still knows about the location of an older version of **PeterBlum.DES.dll** and wanted to copy it into your `\bin` folder. Since you have the latest version in the `\bin` folder, nothing was changed.

Sometimes you can remove this warning with these steps:

1. Open the Solution Explorer
2. Right click on the web application project node.
3. Choose **Properties** from the context menu.
4. Switch to the **Reference Paths** view.
5. If you see a file path to the older version of **Peter's Data Entry Suite**, remove it.

The page generates this error: "Specified cast is not valid." (System.InvalidCastException)

Look at the ASP.NET Declarative Syntax of any Peter's Data Entry Suite control for a child tag that starts with `<PeterBlum.DES.Classname>`. Replace the "PeterBlum.DES" with "des:" or whatever the tag prefix is for the PeterBlum.DES assembly in the `<@ Register>`.

Example

Original

```
<des:RequiredTextValidator [properties]>
  <ErrorFormatterContainer>
    <PeterBlum.DES.TextErrorFormatter Display="Dynamic">
```

```
</PeterBlum.DES.TextErrorFormatter>  
</ErrorFormatterContainer>  
</des:RequiredTextValidator>
```

Corrected (when `<@ Register tagPrefix="des">`):

```
<des:RequiredTextValidator [properties]>  
  <ErrorFormatterContainer>  
    <des:TextErrorFormatter Display="Dynamic">  
    </des:TextErrorFormatter>  
  </ErrorFormatterContainer>  
</des:RequiredTextValidator>
```

The page generates this error: “Security Exception: The application attempted to perform an operation not allowed by the security policy.”

You are running in a Partial Trust Environment. See “Installing into a Partial Trust Environment” in the **Installation Guide**.

The Type attribute refers to the type [typename]

- Confirm that the assembly associated with the “Type” or identified in the error message is in this web application’s **\bin** folder or in the Global Assembly Cache.
- You will get this error if the **PeterBlum.DES.dll** is an earlier version than what was compiled into the other assembly. For example, if you receive SuperDESClasses.dll and it was compiled with **PeterBlum.DES.dll** version 4.0.2.5000 and you have **PeterBlum.DES.dll** version 4.0.1.5000 installed, this error will happen. The solution is to retrieve an update. Go to <http://www.PeterBlum.com/DES/Home.aspx>.

What to do when you get version errors

- If the assembly is any from DES (they all start with “PeterBlum.DES”), run the **Web Application Updater** utility with the option **Update a Web Application (service release)**.
- If you can compile the assembly, recompile it with the same **PeterBlum.DES.dll** that is in use in your web application.
- For any other assembly, open your **web.config** file and locate an `<assemblyBinding>` tag. If any are found, they map a specific assembly in the `<assemblyIdentity>` tag to a desired version number for that assembly.
- Correct or add its `<assemblyBinding>` tag.
- Make sure the `<configuration>` tag of the file does *not* have the `xmlns=` attribute.

Runtime Problems

Controls don't function correctly after an AJAX callback updates the page

You have not set up these controls to correctly use the DES AJAX features. See "Using These Controls with AJAX".

An alert appears with "Page is Loading" while interacting with a DES control

There are two reasons to see an alert with "Page is loading. Please wait".

1. The page is actually still loading. The textboxes use JavaScript to manage keystrokes and that code is initialized near the end of the page load. A slow modem or huge page may be the cause. The message is communicating the issue. The keystroke will be ignored.
2. There was a JavaScript error that occurred as the page loaded and happened before any of DES's page initialization code. See "Handling JavaScript errors". This kind of error should only happen during development and initial testing of a deployed application. Once the deployed application is tested, the alert is due to a slow page load.

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".
- 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".
- If the page is called from `Server.Transfer()`, you must add some code to the original and destination pages. See "Using Server.Transfer".

A property that was set up programmatically is not automatically restored on postback

See "The ViewState and Preserving Properties forPostBack".

When anything is popped up, it appears under some other fields already on the page

When you use absolute positioning for fields, such as by using the `GridLayout` setting on the page in Visual Studio, you can set up a layering effect. The HTML style sheet attribute "ZIndex" controls the layering. The lower the value, the lower the layering.

Z-Index layering is a bit tricky when your HTML is contained within an absolutely positioned `<DIV>`, `<TABLE>` or other container. The container itself can have a z-index that is relative to its peers. Any HTML that it contains is sandwiched between the container and the next highest peer container. Suppose you have the following HTML:

```
<div style='position:absolute;left:0;top:0;width=100;height=200;z-index:100'>
  <span id=span1 style='position:absolute;left:0;top:0;z-index:10'>Span 1</span>
  <span id=span2 style='position:absolute;left:0;top:20;z-index:20'>Span 2</span>
</div>
<div style='position:absolute;left:0;top:0;width=100;height=200;z-index:200'>
  <span id=span3 style='position:absolute;left:0;top:0;z-index:1'>Span 3</span>
  <span id=span4 style='position:absolute;left:0;top:20;z-index:2'>Span 4</span>
</div>
```

Span 1 is lower than Span 2. Span 3 is lower than Span 4. Everything in the first `<DIV>` is lower than everything in the second one because of the z-index on each `<DIV>`. If the `PopupCalendar` was in the first `<DIV>`, it would also be under everything in the second `<DIV>`.

Any DES pops up sets its `Zindex` attribute to values starting at 30000. To fix any controls that overlap these popups, reduce their z-index attributes to be below 30000. Set each absolutely positioned container's z-index to be on the same layer or a lower z-index than the one containing the popup control.

Images and style sheets do not load when the site uses Forms Authentication

Graphic images do not appear and DES style sheets are not loaded when using Forms Authentication for users who are not authenticated (such as are on the login page). This happens when **[Web App Folder]\DES** folder is included within Forms Authentication. It must be excluded.

In the <configuration> section of your **web.config** file, add this to exclude the **DES** folder.

```
<location path="DES">
  <system.web>
    <authorization>
      <allow users="*" />
    </authorization>
  </system.web>
</location>
```

Turning of Gzip/Deflate Compression

When using the DESGetFiles.aspx HttpHandler (the default for ASP.NET 2 and higher users), browsers will receive compressed style sheet and script data using either gzip or deflate compression. If this is causing problems, you can disable this feature.

Add this key to the <appSettings> section of **web.config**:

```
<add key="DES_NoResponseCompression" value="" />
```

Design Mode Problems

Design Mode shows "Error Creating Control" instead of the control

When you view the control in design mode, sometimes you will see the following:

Error Creating Control - Control ID 

Here are several causes of this: incorrectly installed software, incorrect property names or values, and bad ASP.NET text. *Visual Studio 2002 and 2003 users:* If you point to the , VS.Net will show you the error in a tooltip.

- If the error is "**The PeterBlum.ADME.dll is not found in the Global Assembly Cache**", you may have forgotten to install this product. It is required. See the topic "Installing ASP.NET Design Mode Extender" in the **Installation Guide**.
- If the error is "**No file is found at DES\DES.config**", you may have forgotten to move DES's files into your web application. See the topic "First Time Installation" in the **Installation Guide**.
- If the error is "**This control cannot be displayed because its tagprefix is not registered in this web form**", you are missing this <@ Register > tag at the top of the web form or user control:


```
<%@ Register TagPrefix="des" Namespace="PeterBlum.DES"
  Assembly="PeterBlum.DES" %>
```
- If the error is "**Parser error: Type 'PeterBlum.DES.class' does not have a property named 'propertyname'**", you have entered the property name incorrectly.
- If the error is "**Specified cast is not valid**", click here.
- For other errors, use the error message for guidance. If you need technical support, please include the error message.

Each control shows a small icon like this:

DES is telling you that there is a problem. Click on the icon to display an alert with details. There are two cases:

- Licensing is not setup for the control. You will not be able to use this control at runtime until licensing is corrected. See the "Installing Licenses" section of the **Installation Guide**. It includes a troubleshooting topic too.
- DES cannot load from its config files in the **DES** folder of your web application. You will not be able to use certain design mode features that are based on the configuration files, but you can work with design mode in general.

When this error is detected, it indicates a problem internal to Visual Studio. VS accesses two copies of the **PeterBlum.DES.dll** assembly, one for drawing in design mode and the other for other operations. These two copies are out of sync (usually different compiles). Restarting your computer usually helps VS reset this. If you have the Clean Solution command in your VS menus, use it first, then restart.

Each control shows a small icon like this:

This icon is intended to help you differentiate its controls from others. In many cases, the DES controls look the same as native ASP.NET controls, such as the TextBox, Button, and Validators. Since you want your forms to use the correct controls – either DES's or the native ones, this is intended to assist you.

If it is excessive, DES provides a setting to diminish its use or turn it off.

1. Open the **Global Settings Editor**.
2. Go to the Visual Effects topic.
3. Use the **ShowDESBmp** property. The InBothFrameworks option shows the icon only in controls that appear in both DES and the ASP.NET controls.

Images do not appear in design mode, but they appear at runtime

When using ASP.NET 1.x, make sure the ASP.NET Design Mode Extender ("ADME") is set up correctly.

- Open the **ADME Settings Editor**. It is usually in the Visual Studio Tools menu or a button on the toolbar
- Make sure there is a project listed for the current web application you are using
- Click the Open button. Review the properties to confirm they are correct. Click OK.
- On the main window, the *Design Mode is working with* field should identify the web application.
- Restart Visual Studio.net

When using ASP.NET 2.0, its design mode feature-set provides a replacement to the ADME utility. It's that replacement that can cause problems. Here are the known cases:

- You are using the ASP.NET 1.x version of the **PeterBlum.DES.dll**. It only works with ADME. Remove the project reference to it and add a reference to the one in the **[DES product folder]\Assemblies\ASPNET 2_0** folder.
- You have set up remote access to the web application. Visual Studio copies files from the remote server locally. It has failed to copy the **DES\Appearance** folder or all of its contents. Copy it manually to the local copy of your web application.

The DES control lacks some properties in the Properties Editor

You may not have a license set up that supports the missing properties. For example, a Button's DisableOnSubmit property is not show if you don't have a license for Peter's Interactive Pages (or the Suite).

Working in Visual Studio's Properties Editor stops transferring properties into the ASP.NET text

If you are using Visual Studio for a while, the Properties Editor may start to malfunction. It will not transfer settings that you've made in it into the ASP.NET text page. (VS.NET has the responsibility of converting the object in the Properties Editor into text following various design mode attributes on the controls.)

If you have any kind of malfunction in the Properties Editor, restart Visual Studio.

Intellisense does not work with these controls in Visual Studio 2002 or 2003

Visual Studio.net 2002 and 2003 do not support Intellisense on custom controls. Developers have found a hack that gives you support. However, it is awkward and likely to require more work than it is worth. You will create an XML file using a third party program. Then for each page that uses these controls, add the namespace= attribute to the <body> tag to point to that XML file. *PeterBlum.com will not provide any additional technical support on this issue.*

1. Get the program Intellisense Generator from BlueVision Software at:
<http://www.bluevisionsoftware.com/WebSite/ProductsAndServicesInfo.aspx?ID=9>
2. Run the program on the PeterBlum.DES.dll and save the file in a location where a URL can reference it such as in your \aspnet_client folder.
3. Update your web form's <body> tag like this:
`<body namespace="http://[URL]">`

Intellisense is missing some properties from these controls in Visual Studio 2005

Visual Studio 2005 caches the Intellisense text it extracts from assemblies. Sometimes this cache is out of date, causing properties and methods introduced after the data was cached to be ignored.

1. Exit Visual Studio
2. In Windows Explorer, go to [documents and settings][user name]\Application Data\Microsoft\Visual Studio\8.0\Reflected Schemas
3. Delete all of the files in that folder.