DES DYNAMIC DATA USER'S GUIDE

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

Click on any of these topics to jump to them:

- What is "Dynamic Data"? How DES enhances ASP.NET Dynamic Data
- Installing Dynamic Data Into Your Web Application
- FAQs Terminology Table Of Contents
- Other assistance ASP.NET Dynamic Data LINQ to SQL ASP.NET Routing

Business logic Overview
- Creating EntityTable Classes Using LINQ to SQL Using ADO.NET Entity Framework
- Using EntityTable MetaData Classes Creating Applying Attributes DataTypeAttributes ValidationAttributes More
- Runtime Customization ICustomizeColumn ICreateValidatorWebControls

Web form development Overview
- Building Applications with PageTemplates Standard PageTemplates Table-specific PageTemplates
- Building Applications without PageTemplates DynamicControls DynamicFields
- PatternTemplates DynamicPattern control DynamicFormView control DynamicListView control NamedStyles Shipped with DES
  Creating DynamicStylesTag control DynamicPartsPatterns control
- DynamicDataManager Adapters
- Other Controls Customizer DynamicLabel DynamicValidator DynamicButtons ColumnValidatorManager DynamicColumnTitle DynamicControlPanel
- Runtime Customization IUICustomizeColumn Customizing Automatic Scaffolding ICustomizeDataBoundControl ICustomizeFieldTemplate

Field Templates
- Overview Creating Setting Properties Classes and Properties
- Shipped with DES Dynamic Data

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

Table of Contents

License Information 23

Platform Support 23

Technical Support and Other Assistance 24
- Troubleshooting Section of this Guide 24
- Developer's Kit 24
- PeterBlum.Com MessageBoard 24
- Getting Product Updates 24
- Technical Support 24

OVERVIEW OF DES DYNAMIC DATA 26

What is "Dynamic Data"? 27
- Advantages 27
- Limitations 27
- Learn More 28

How Dynamic Data changes how you build web forms 29
- Separation of Concerns 29
- Developers can play Different Roles 30
- Structure of a Dynamic Data driven web form 31
  - When in Edit or Insert modes 32

The Elements of Dynamic Data 33
- Classes that Represent your Data 33
- User Interface Elements 35
  - Working with Data Fields 35
  - Working with DataBound controls 36
  - Controls used within DataBound controls and PatternTemplates 38
  - Other Controls 39
- Attributes Assigned to the Classes Representing Your Database 40
- Files and Folders 44

How DES enhances ASP.NET Dynamic Data 45
- Summary of major features 45
  - Business rules 45
  - Field Templates 46
  - Webform Development 46
- Controls 47
- Field Templates 49
- PatternTemplates 50
- PageTemplates 51
- Attributes 52

FAQS 53

INSTALLING DYNAMIC DATA INTO YOUR WEB APPLICATION 55

How will you use Dynamic Data? 55

Prerequisites for using DES Dynamic Data 56

Adding ASP.NET Dynamic Data 57

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

Using Visual Studio – For new applications only 58
Using Visual Studio – For existing applications only 59
Not Using Visual Studio 62

Adding DES Dynamic Data 64
Steps to add DES Dynamic Data support 65

Initializing Dynamic Data at application startup 67
Create the DESMetaModel object 68
Make these edits 68
Set up ASP.NET URL Routing 69
Mapping PageTemplates to Actions 69
Using Versatile DataSources 70
When using ADO.NET Entity Framework 70
When using POCODataSource 70
When Missing the Global.asax File 71

OVERVIEW: BUSINESS LOGIC ON YOUR TABLES 72

Choosing a Technology for your Business Logic 73
Recommendation 73
Query ("read") action on Products Entity class 74

CREATING ENTITYTABLE CLASSES 75

Using LINQ to SQL 76

Using ADO.NET Entity Framework 78

Using Versatile DataSources 79

Registering the DataContext with Dynamic Data 80

DEFINING BUSINESS LOGIC ON EACH ENTITYTABLE 82

Overview 83
Steps for defining your business logic 85

Creating the EntityTable Metadata Classes 90

Applying Attributes To Columns 92

DataTypeAttributes: The DataTypes used by your Business logic 93
Mapping DataType to its DataTypeAttribute 94
DES.DataTypeAttribute 95
System.ComponentModel.DataAnnotations.DataType 96
PeterBlum.DES.DynamicData.ExtendedDataType 98
String 99
How to determine the DataType from within a Field Template 100
DES.IntegerDataTypeAttribute 101
Properties 101
DES.DecimalDataTypeAttribute 102
Properties 102
DES.CurrencyDataTypeAttribute 103
Properties 103
DES.PercentDataTypeAttribute 104

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

Properties 104
DES.IntegerMeasurementDataTypeAttribute 105
Properties 105
DES.DecimalMeasurementDataTypeAttribute 107
Properties 107
DES.DateDataTypeAttribute 109
Properties 109
DES.TimeOfDayDataTypeAttribute 110
Properties 110
DES.DurationDataTypeAttribute 112
Properties 112
DES.AnniversaryDataTypeAttribute 114
Properties 114
DES.MonthYearDataTypeAttribute 115
Properties 115
DES.EnumeratedAttribute 116
Using this Attribute 116
Properties 117
Using the CustomizeColumn Method 120
DES.EnumeratedStringAttribute 122
Using this Attribute 122
Properties 123
Using the CustomizeColumn Method 126
DES.UrlDataTypeAttribute 128
Properties 128
DES.EmailAddressDataTypeAttribute 131
Properties 131
Providing additional validation of email addresses 134
DES.UrlAsImageDataTypeAttribute 135
Using this Attribute 135
Properties 135
DES.DbImageDataTypeAttribute 137
Using this Attribute 137
Properties 137

ValidationAttributes: Applying Constraints on Columns 140
Standard types vs DES types 141
ErrorMessages on ValidationAttributes 142
ErrorMessage Properties found on DES ValidationAttributes 142
How Field Templates utilize the ValidationAttributes 144
Step by Step instructions 144
The types of ValidationAttributes supplied with DES Dynamic Data 145
Using the CustomizeColumn Method 146
Method: GetExistingValidationAttribute 146
Method: GetValidationAttribute 150
Method: RemoveValidationAttribute 152

Writing server side only Validation Code within your EntityTable Classes 153
Using ASP.NET 4: The CustomValidationAttribute 154
Understanding the CustomValidationAttribute declaration 155
Understanding the method definition 156
Using ASP.NET 3.5 158

Invoking Validation prior to saving 160
Working Data Access Objects from Versatile DataSources 160
Working with the Domain object from Domain Services 160
Working with LINQ to SQL and no Data Access Object 161
Working with Entity Framework and no Data Access Object 161

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

DES.RequiredAttribute 162
Using this Attribute 162
Properties 162
DES.RangeAttribute 163
Using this Attribute 163
Properties 163
DES.CompareToValueAttribute 164
Using this Attribute 164
Properties 164
DES.RegularExpressionAttribute 165
Using this Attribute 165
Properties 165
DES.StringLengthAttribute 167
Using this Attribute 167
Properties 167
DES.CompareTwoColumnsAttribute 168
Using this Attribute 168
Properties 168
DES.DifferenceAttribute 169
Using this Attribute 169
Properties 169
DES.StringListAttribute 171
Using this Attribute 171
Properties 171
DES.InjectionSecurityAttribute 173
Using this Attribute 173
Properties 173
DES.CharacterSetAttribute 176
Using this Attribute 176
Properties 176
DES.CreditCardNumberAttribute 179
Using this Attribute 179
Properties 179
DES.ABARoutingNumberAttribute 180
Using this Attribute 180
Properties 180
ValidationAttributes that define the DataTypeCheckValidator 181

DependencyAttributes: Relationships between columns 182
Using DependencyAttributes 182
Using the Column's Metadata 183
Using the CustomizeColumn Method 184
Method: GetDependencyAttribute 184
Method: CreateRequiredDependencyAttribute 186
Method: CreateConditionDependencyAttribute 188
Method: RemoveDependencyAttribute 189
RequiredDependencyAttribute 190
Properties 190
ConditionDependencyAttribute 192
Properties 193

UIHintAttribute: Customizing the Field Template used 194
Selecting an alternative Field Template 194
Applying the UIHintAttribute to select an alternative Field Template 194
Setting properties on Field Templates 196
PropertyList Syntax 197
When creating DES.UIHintAttribute programmatically 197

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

DisplayNameAttribute: Friendly column names 199
Using the DisplayNameAttribute 199
Properties 199

DescriptionAttribute: A Hint for the user 201
Using the DescriptionAttribute 201
Properties 201
Using the CustomizeColumn Method 203

DefaultValueAttribute: Initial value when creating a new record 205
Using the CustomizeColumn Method 206

DisplayFormatAttribute: Rules to convert from value to a string 207
Properties 207
Using the CustomizeColumn Method 208

ScaffoldColumnAttribute: Determining if Scaffolding adds a Column 209
Using the DES.ScaffoldColumnAttribute 209
How Scaffolding works when there is no ScaffoldColumnAttribute specified 209
Syntax 210
Properties of the DES.ScaffoldColumnAttribute 211
Properties to include or exclude columns 211
Properties for positioning 211

SortableColumnAttribute: Determining sorting rules 212
Using the SortableColumnAttribute 212
Properties 212

CategoryAttribute: Assigning Categories to Data Fields 213
Using the CategoryAttribute 213
Properties 213

EditableAttribute: Prevent editing on certain columns 214
Using The EditableAttribute 214
Properties 214
Syntax 214

ColumnRestrictionAttribute: Deny access to columns by Role 215
Using the DES.ColumnRestrictionAttribute 215
Properties 216

CacheListAttribute: Caching Lists for Enumerated and ForeignKeys 217
Using the CacheListAttribute 217
Properties 217
Using the CustomizeColumn Method 219

Customizing Metadata At Runtime 221
Understanding the ModifiableColumn class 222
ModifiableColumn properties that you should not modify 222
ModifiableColumn properties that you can modify 222
Implementing the ICustomizeColumn Interface 225
Implementing the IUICustomizeColumn Interface 227
Where to implement IUICustomizeColumn 227
Adding the IUICustomizeColumn Interface 228
The GetUICustomizeColumnBehavior method 229
The CustomizeColumn method 230

Applying Attributes To EntityTables 232

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

DisplayNameAttribute: Friendly table names **233**
Using the DisplayNameAttribute 233

DisplayColumnAttribute: Selecting a column name for foreign keys **234**
Specifying an alternative column name 234
Customizing the data shown 234
Sorting by a different column 234

ScaffoldTableAttribute: Determining if a table appears in the UI **236**
Using the ScaffoldTableAttribute 236
Properties 236

TableRestrictionAttribute: Deny access to tables by Role **237**
Using the DES.TableRestrictionAttribute 237
Properties 238

Defining Validators at Runtime: ICreateValidatorWebControls Interface **239**
Adding the ICreateValidatorWebControls Interface 239
The CreateValidatorWebControls method 240

Creating Calculated and Custom Columns **242**

Creating Instances of the EntityTable classes **243**
The IEntityTypeCreator Interface 243
Where to implement IEntityTypeCreator 243
Adding the IEntityTypeCreator Interface 244
The CreateEntityType method 245

SETUP SECURITY BY USER ROLES **246**

Features **246**

Global setup for Security Roles **247**
Setup when using FormsAuthentication 247
Subclassing to define restrictions by role 250
Setup using an alternative RestrictionManager 252
Blocking restricted URLs 253

Customizing the User Interface to Respect Security **254**
Securing pages with DataBound controls 255
Throw an exception 255
Redirect to another page 255
Replace the DataBound control with alternative HTML 256
Change the DataSource's query 257
Hide tables listed in the Entry Point Page 258
Showing an alternative interface with a DES.DynamicControl 259
Edit mode reverts to ReadOnly mode 259
ReadOnly mode shows a User Control 259
Customizing Automatic Scaffolding 260

Using the RestrictionManager in your own code **261**
CanTakeActionOnTable method 262
IdentifyTableRestrictions method 263
CanTakeActionOnColumn method 264
IdentifyColumnRestrictions method 265
HasRole method 266

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

WEB FORM DEVELOPMENT OVERVIEW 268

Several ways to work: Choosing the right technique 269

BUILDING APPLICATIONS WITH PAGETEMPLATES 273

Automatic Scaffolding Overview 274
PageTemplates and Automatic Scaffolding 275
Overcoming the limitations of Automatic Scaffolding 276
Tools to customize Automatic Scaffolding 276

Files in the PageTemplates System 277

Steps for using PageTemplates 278

Customizing the standard PageTemplates 280
Before you start 280
Elements of a PageTemplate 281
Setting Properties on your Data Source 285
Setting Properties on your DataBound control 286
Properties 286
Adding Buttons to your DataBound control 287
Automatic generation: Let the DataBound control add them for you 288
Declaring the buttons within the list of fields 290
Using actual Button controls 292
Redirecting to another PageTemplate on a button click 296
Using the GetActionPath method 296
Redirect using a Hyperlink 297

Creating Table-Specific PageTemplates 298
Customizing Automatic Scaffolding 298
Using the CustomPages folder 298
Steps to using the CustomPages folder 298
Using DynamicListView and DynamicFormView Controls 299
Using ListView and FormView Controls 301
Switching from GridView or DetailsView 301
Switching from DynamicListView or DynamicFormView 301
Using GridView and DetailsView Controls 303

Customizing Automatic Scaffolding 304
Different Approaches for Customization 305
Customizing with native Dynamic Data 305
Customizing with DES Dynamic Data 306
Customizing your own way 307
The IPatternTemplateFieldGeneratorGuidance interface 308
Where to implement IPatternTemplateFieldGeneratorGuidance 309
Adding the IPatternTemplateFieldGeneratorGuidance Interface 310
IPatternTemplateFieldGeneratorGuidance.BeforeGeneration method 311
IPatternTemplateFieldGeneratorGuidance.CreateField method 315
IPatternTemplateFieldGeneratorGuidance.AfterGeneration method 319
The IDataControlFieldGeneratorGuidance interface 321
Where to implement IDataControlFieldGeneratorGuidance 322
Adding the IDataControlFieldGeneratorGuidance Interface 323
IDataControlFieldGeneratorGuidance.BeforeGeneration method 324
IDataControlFieldGeneratorGuidance.CreateField method 328
IDataControlFieldGeneratorGuidance.AfterGeneration method 332
PeterBlum.DES.DynamicData.FieldGenerationArgs 334
FieldGenerationArgs Properties 334

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

AddColumn() method 337
AddField() method 339
GetFieldByColumnName() method 342
HideFieldByColumnName() method 344
MoveFieldByColumnName() method 345
Creating your own FieldGenerator Class 347

Customizing the Master Page 348
Elements of a Master Page 348
Renaming the Master Page 350
Change the appearance of the Master Page 351
Change the style sheet file 352
Handling Conflicts between style sheets and web controls 352
Selecting between several Master Pages on a PageTemplate 354

Customizing the Entry Point Page 355
Elements of an Entry Point Page 355
Renaming the Entry Point Page 357
Going to a specific table 358

Changing the Default paths for PageTemplates 359
Steps for changing paths 359

Setting up the DataBound Control at Runtime 361
Implementing the ICustomizeDataBoundControl Interface 362
Where to implement ICustomizeDataBoundControl 362
Adding the ICustomizeDataBoundControl Interface 363
ICustomizeDataBoundControl.CustomizeDataBoundControl method 364
Util.GetTableName method 366
Util.GetFieldTemplate method 367
Util.GetTable method 369

BUILDING APPLICATIONS WITHOUT PAGETEMPLATES 370

Steps for adding Dynamic Data Features into your forms 371

PATTERN TEMPLATES: REUSABLE FORMATTING FOR DYNAMICCONTROLS 381

Overview 382
Let's start with an example 383
Switching to a PatternTemplate: The DynamicPattern control 384
Named Styles: Your pages control the formatting 385
The NamedStyle class 385
The DynamicStylesTag control 386
Other ways to apply Named Styles 387
Merging Style Sheet Classes 387
The Powerful DynamicListView and DynamicFormView controls 388
Location of your PatternTemplate files 389
PatternTemplates shipped with DES Dynamic Data 390
Style sheets 391

The DynamicPattern Control 392
Using the DynamicPattern Control 393
Properties that should always be considered 394
Defining ItemInPattern objects in the ItemsInPattern collection 396
Defining the Data Field names shown by the PatternTemplate 397
Buttons in PatternTemplates 400

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

Passing text into PatternTemplates 401
Using PatternTemplates to output content without data fields 402
Adding HTML and other web controls between data fields 403
Using Named Styles to adjust the formatting 404
Passing properties to the Field Templates 405
Passing properties to be used by the PatternTemplate's own code 406
Adding a DynamicPattern Control 407
Properties for the DynamicPattern Control 410
Template Lookup Properties 411
Data Properties 412
Appearance Properties 413
PartsPatternTemplate Properties 414
Default NamedParts Properties 416
Scaffold Caching Properties 417
Behavior Properties 418
Methods for the DynamicPattern Control 419
AddDataField() method 420
AddDataFieldList() method 421
AddButtons() method 422
AddScaffolding() method 423
FindByID() method 424
FindByDataField() method 425
GetAttributeAsInteger() method 426
GetAttributeAsBoolean() method 427
GetAttributeAsString() method 428
GetAttributeAsUnit() method 429
GetAttributeAsNative() method 430
SetAttributeAsNative() method 431

The DataFieldInPattern Class 432
Using the DataFieldInPattern class 433
Basic setup: Always assign data field name 433
Use a NamedPart in a PartsPatternTemplate 433
Specify a UIHint to identify the Field Template 435
Supply alternative values for properties on the Field Template 435
Customizing the text for a DynamicLabel 436
Specify which DynamicControl and DynamicLabel gets assigned the data field 436
Properties of the DataFieldInPattern class 437
Data Properties 437
Behavior Properties 439
Field Template Properties 441
Constructors for the DataFieldInPattern class 442

The DataFieldListInPattern Class 444
Using the DataFieldListInPattern Class 445
Properties of the DataFieldListInPattern class 447
Key Properties 448
Behavior Properties 449
Methods for the DataFieldListInPattern class 450
AddDataField() method 450
AddButtons() method 451
FindByID() method 452
FindByDataField() method 453
Constructors for the DataFieldListInPattern class 454

The ScaffoldInPattern Class 455
Using the ScaffoldInPattern class 456
Properties of the ScaffoldInPattern class 457

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

Constructors for the ScaffoldInPattern class....458

The ButtonsInPattern Class....459
Using the ButtonsInPattern class....460
Properties of the ButtonsInPattern class....461
Constructors for the ButtonsInPattern class....462

The LabelInPattern Class....463
Using the LabelInPattern class....464
Properties of the LabelInPattern class....465
Constructors for the LabelInPattern class....467

The NoDataFieldsInPattern Class....468
Using the NoDataFieldsInPattern class....469
Properties of the NoDataFieldsInPattern class....470
Constructors for the NoDataFieldsInPattern class....471

The TemplateNotInPattern Class....472
Using TemplateNotInPattern....473
Properties of the TemplateNotInPattern class....474
Constructors for the TempateNotInPattern class....476

The NamedStyle Class....477
Properties on the NamedStyle class....478
Identity Properties....478
CSS Properties....480
Other HTML Attribute Properties....482

The DynamicListView Control....483
Using the DynamicListView Class....484
Getting Started....485
Comparing to ListView....486
How to use the PartsPatternTemplate with a DynamicListView....487
Parts Overview for PartsPatternTemplate when used with Lists....488
Establishing a DataSource....490
Defining a list of data fields....492
Changing the appearance....493
Passing properties to be used by the PatternTemplate's own code....499
Offering Paging....500
Grouping records to provide a tiled interface....501
Adding the DynamicListView Class....503
Properties of the DynamicListView Class....506
Data Properties....507
PatternTemplate Properties....508
Default NamedParts Properties....509
Overall appearance Properties....510
Header and Footer Properties....511
Grouping Properties....513
Template Properties....514
Scaffold Caching Properties....515
Behavior Properties....516
Properties used when programming....518

The DynamicFormView Control....519
Using the DynamicFormView Class....520
Comparing to FormView....521
How to use the PartsPatternTemplate with a DynamicFormView....522
Parts Overview for PartsPatternTemplate when used with Single Records....523

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

Establishing a DataSource 525
Defining a list of data fields 526
Changing the appearance 527
Passing properties to be used by the PatternTemplate's own code 532
Offering Paging 533
Adding the DynamicFormView Class 534
Properties of the DynamicFormView Class 537
Data Properties 538
PatternTemplate Properties 539
Default NamedParts Properties 540
Appearance Properties 541
Template Properties 543
Scaffold Caching Properties 544
Behavior Properties 545

Using the PatternTemplates shipped with DES Dynamic Data 546
Using DetailsView.ascx PatternTemplate 547
Example: DynamicPattern in FormView 548
Example: DynamicPattern with quick setup 548
Example: DynamicFormView 548
NamedStyles on the DetailsView PatternTemplate 549
NamedParts on the DetailsView PatternTemplate 550
Properties on the DetailsView PatternTemplate 551
Using GridView.ascx PatternTemplate 552
Example: DynamicPattern in ListView 553
Example: DynamicListView 554
NamedStyles on the GridView PatternTemplate 555
NamedParts on the GridView PatternTemplate 556
Using ListOfDetailViews.ascx PatternTemplate 557
Example: DynamicPattern in ListView 558
Example: ListView in using Groups 559
Example: DynamicListView 560
Example: DynamicListView using Groups 560
NamedStyles on the ListOfDetailViews PatternTemplate 561
NamedParts on the ListOfDetailViews PatternTemplate 562
Using TwoColumnTableRow.ascx PatternTemplate 563
NamedStyles on the TwoColumnTableRow PatternTemplate 563
Using TwoStacked.ascx PatternTemplate 564
NamedStyles on the TwoStacked PatternTemplate 564
Properties on the TwoStacked PatternTemplate 564
Using DetailsViewButtons.ascx and DetailsViewLinkButtons.ascx PatternTemplates 565
NamedStyles on these PatternTemplates 566
Using ButtonsForList.ascx and LinkButtonsForList.ascx PatternTemplates 567
NamedStyles on these PatternTemplates 568
Using ButtonsForSingle.ascx and LinkButtonsForSingle.ascx PatternTemplates 569
NamedStyles on these PatternTemplates 570

Customizing and creating your own PatternTemplates 571
The format of PatternTemplate files 572
Creating your PatternTemplate with the DataFieldsPatternTemplate class 573
Steps for creating a DataFieldsPatternTemplate 573
Creating your PatternTemplate with the PartsPatternTemplate class 579
Steps for creating a PartsPatternTemplate 579
Overriding the PartsPatternTemplate.OnPartCreating() method 587
The PeterBlum.DES.DynamicData.PartCreatingEventArgs class 589
Properties of PartCreatingEventArgs 590
GetAttributeAsNative method 592
GetAttributeAsString method 592

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

RegisterDynamicControl method 593
RegisterDynamicLabel method 593
CreateFromPart method 594
Overriding the PartsPatternTemplate.OnPartCreated() method 595
The PeterBlum.DES.DynamicData.PartCreatedEventArgs class 597
Properties of PartCreatedEventArgs 598
FindDynamicControl method 600
FindDynamicLabel method 600
UpdateStyleOnDynamicControls method 601
UpdateStyleOnDynamicLabels method 602
Creating your PatternTemplate with the DynamicButtonsPatternTemplate class 603
Steps for creating a DynamicButtonsPatternTemplate 603
Creating your PatternTemplate with the NoDataFieldsPatternTemplate class 612
Steps for creating a NoDataFieldsPatternTemplate 612
Supporting Named Styles within your PatternTemplates 618
The PatternTemplate.GetTagAttributes method 619
The PatternTemplate.ApplyNamedStyle method 621
The PatternTemplate.GetCssClassFromNamedStyle method 623
The PatternTemplate.GetNamedStyle method 624

The DynamicPartsPatterns Control 625
Parts Overview 626
Selecting the Parts from the DynamicPattern control 627
NamedParts Overview 628
Using NamedParts from DynamicListView, DynamicFormView and DynamicPattern controls 629
Properties of the DynamicPartsPatterns Control 630
Using the Container Part 631
ASP.NET Markup 631
Using this Part within a ListView control 631
Using this Part within a FormView control 632
Using the DataRow Part 633
ASP.NET Markup 633
Example: DynamicControls within DataRow; DataCells not used 634
Example: DataCells for DynamicControls; DataRow provides the Table Row tag 634
Example: DataCells for DynamicControls; DataRow provides a header 635
Using this Part within a ListView control 635
Using this Part within a FormView control 636
Using the DataCell Part 637
ASP.NET Markup 637
Example: DataCells for DynamicControls; DataRow provides the Table Row tag 638
Example: DataCells for DynamicControls; DataRow provides a header 638
Using this Part within a ListView control 639
Using this Part within a FormView control 640
Using the Buttons Part 641
ASP.NET Markup 641
Example 642
Using this Part within a ListView control 643
Using this Part within a FormView control 644
Using the HeaderRow Part 645
ASP.NET Markup 645
Example: DynamicColumnTitles within HeaderRow; HeaderCells not used 646
Example: HeaderCells for DynamicControls; HeaderRow provides the Table Row tag 646
Example: HeaderCells for DynamicControls; HeaderRow provides a header 647
Using this Part within a ListView control 647
Using this Part within a FormView control 647
Using the HeaderCell Part 648
ASP.NET Markup 648
Example: HeaderCells for DynamicControls; HeaderRow provides the Table Row tag 649

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

Example: HeaderCells for DynamicControls; HeaderRow provides a header 649
Using this Part within a ListView control....650
Using this Part within a FormView control 650
Using the HeaderButtons Part....651
ASP.NET Markup....651
Example 652
Using this Part within a ListView control....653
Using this Part within a FormView control 653
Using the FooterRow Part....654
ASP.NET Markup....654
Example: DynamicColumnTitles within FooterRow; FooterCells not used....655
Example: FooterCells for DynamicControls; FooterRow provides the Table Row tag....655
Example: FooterCells for DynamicControls; FooterRow provides a header....656
Using this Part within a ListView control....656
Using this Part within a FormView control 656
Using the FooterCell Part 657
ASP.NET Markup....657
Example: FooterCells for DynamicControls; FooterRow provides the Table Row tag....658
Example: FooterCells for DynamicControls; FooterRow provides a header....658
Using this Part within a ListView control....658
Using this Part within a FormView control 658
Using the FooterButtons Part....659
ASP.NET Markup....659
Example 660
Using this Part within a ListView control....660
Using this Part within a FormView control 660
Using the NoDataToShow Part....661
ASP.NET Markup....661
Example: Using Message property 662
Example: Using Template property 662
Using this Part within a ListView control....662
Using this Part within a FormView control 662
Using the DataRowSeparator Part 663
ASP.NET Markup....663
Example: DataRowSeparatorCells not used 664
Example: DataRowSeparatorCells are used....664
Example: Both DataRowSeparator and DataRowSeparatorCells provide content 664
Using this Part within a ListView control....664
Using this Part within a FormView control 664
Using the DataRowSeparatorCell Part....665
ASP.NET Markup....665
Example: DataRowSeparatorCells not used 666
Example: DataRowSeparatorCells are used....666
Example: Both DataRowSeparator and DataRowSeparatorCells provide content 666
Using this Part within a ListView control....666
Using this Part within a FormView control 666
Using the GroupContainer Part....667
ASP.NET Markup....667
Using this Part within a ListView control....667
Using this Part within a FormView control 668
Using this Part within a DynamicListView control 668
Using the GroupHeader Part....669
ASP.NET Markup....669
Example 670
Using this Part within a ListView control....670
Using this Part within a FormView control 670
Using this Part within a DynamicListView control 670
Using the GroupFooter Part....671

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

ASP.NET Markup....671
Example....672
Using this Part within a ListView control....672
Using this Part within a FormView control....672
Using this Part within a DynamicListView control....672
Using the GroupSeparator Part....673
ASP.NET Markup....673
Example....674
Using this Part within a ListView control....674
Using this Part within a FormView control....674
Using this Part within a DynamicListView control....674
Using the EmptyItem Part....675
ASP.NET Markup....675
Example: EmptyItemCells not used....676
Example: EmptyItemCells are used....676
Example: Both EmptyItem and EmptyItemCells provide content....676
Using this Part within a ListView control....676
Using this Part within a FormView control....676
Using the EmptyItemCell Part....677
ASP.NET Markup....677
Example: EmptyItemCells not used....678
Example: EmptyItemCells are used....678
Example: Both EmptyItem and EmptyItemCells provide content....678
Using this Part within a ListView control....678
Using this Part within a FormView control....678
Properties on each Part....679
Container Tag Properties....679
Inner content Properties....682
NamedParts Properties....684
Behavior Properties....685

The DynamicStylesTag Control....686
Using the DynamicStylesTag Control....687
Creating Named Styles in PatternTemplates....687
Properties of the DynamicStylesTag control....688
Main properties....688
Appearance properties....690
Behavior properties....691

DYNAMICDATAMANAGER CONTROL AND ADAPTERS....692

Features of the DES DynamicDataManager control....693

Using the DynamicDataManager control....694
Overview of DataBound Control Adapters....694

Properties on DES DynamicDataManager....696

Methods on DES DynamicDataManager....699
ChangeMode method....699
GetValidationGroupName method....700
GetDataBoundControlAdapter method....701
SetupButtonsOnClientSide method....702
GetCustomizer method....703
RunActionForCommand method....704
ExtractValues method....705

The BaseDataBoundControlAdapter class....706

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

Properties common to all DataBound Control Adapters....707
Connections to Controls on the Page....707
Behavior Properties....709
Automatic Scaffolding Properties....710
Setting up Buttons Properties....712
Command Action Properties....713
Additional Properties....714
Properties on DetailsViewAdapter, FormViewAdapter, and DynamicFormViewAdapter....715
Command Action Properties....715
Useful Methods on DataBound Control Adapters....716

Using Dynamic Data without the normal DataSource controls....720
The ObjectDataSource control....721
An Entity class....722
The EnableDynamicData method....722
The BaseDataControlAdapter.EnableDynamicData property....723
A DataTable....724

DES.DYNAMICCONTROL: THE HEART OF DYNAMIC DATA....725

Features of DES.DynamicControl....725

Using DES.DynamicControl....726
Where to add DynamicControls....726
Assigning the DataField....726
Selecting the Field Template....727
Supply alternative values for properties on the Field Template....728
Syntax of these properties....728
When creating DES.DynamicControl programmatically....729

Adding a DES.DynamicControl....730

Properties on DES.DynamicControl....732
Data Properties....732
Field Template Properties....732
Data Formatting Properties....733
Other Properties....733

DES.DYNAMICFIELD: USING DYNAMIC DATA IN GRIDVIEW AND DETAILSVIEW....734

Features of DES.DynamicField....734

Using DES.DynamicField....735
Assigning the DataField....735
Selecting the Field Template....736
Supply alternative values for properties on the Field Template....737
Syntax of these properties....737
When creating DES.DynamicField programmatically....738

Properties on DES.DynamicField....739
Data Properties....739
Data Formatting Properties....739
Field Template Properties....740
Appearance Properties....741
Behavior Properties....742

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

CUSTOMIZER CONTROL 743

Using the Customizer Control 744
Specifying the Customizer on the DataBound Control Adapter 744

Properties on the Customizer Control 745
Date and Time Field Template Properties 745
Numeric Field Template Properties 748
Text Field Template Properties 752
Button Properties 754
Validation Properties 756
ForeignKey Field Template Properties 760
Miscellaneous Properties 761

COLUMNVALIDATORMANAGER CONTROL 763

Setting Appearance properties 764

Modifying Validators after they have been created 765

DYNAMICVALIDATOR CONTROL 766

THE DYNAMICLABEL CONTROL 767

Using the DynamicLabel Control 768
Establishing Metadata on a column 768
Establishing the DataField name 769
Choosing a source for the column's metadata 770
Formatting the text 771
Enclosing the label in other HTML formatting 772

Properties on the DynamicLabel Control 773
Source of data Properties 773
Textual Content Properties 774
Additional formatting properties 776

THE DYNAMICBUTTONS CONTROL 777

Using the DynamicButtons Control 778
Choosing the Appearance: from Properties or Templates 778
Using the Layout Properties 779
Using Templates 780

Adding a DynamicButtons Control 784

Properties for the DynamicButtons Control 786
Button Visibility Rules properties 787
Buttons without Templates properties 789
Button Appearance properties 790
Button Layout properties 791
New Record Button properties 792
Edit Record Button properties 793
Delete Record Button properties 794
Details Button properties 795
Select a Row Button properties 796
Save New Record Button properties 797

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

Cancel New Record Button properties....798
Save Edit Record Button properties....799
Cancel Edit Record Button properties....800
Button Templates properties....801
Container Appearance properties....802
Behavior Properties....803
ButtonsCreating Event Handler....804
ButtonsCreated Event Handler....805

The VisibilityByCommand Control....807

THE DYNAMICCOLUMNTITLE CONTROL....808

Using the DynamicColumnTitle control....809
Establishing Sorting on a Data Field....809
Establishing the DataField name....809
Choosing a source for the column's metadata....810
Formatting the text....811
Sorting Interface Appearance....812

Properties of the DynamicColumnTitle control....813
Sorting Interface Properties....813
Behavior Properties....814

THE DYNAMICCONTROLPANEL CONTROL – THE ENHANCED DATAPAGER....815

Features....817

Using the DynamicControlPanel....818
Connecting to the DataBound control....819
Fields – Subclasses of BaseDynamicControlPanelField....820
Using the NextPreviousPagerField....822
Using the PreviousPagerField and NextPagerField....823
Using the NumericPagerField....824
Using the IncrementExtendedNumericPagerField....826
Using the SpreadExtendedNumericPagerField....827
Using the JumpByExtendedNumericPagerField....828
Using the InfoPagerField....829
Using the DropDownPageSizesField and LinksPageSizesField....830
Using the DropDownSorterField and LinksSorterField....831
Using the CommandButtonField....833
Using the NewCommandButtonField....834
Using the DynamicButtonsField....835
Using the TemplateControlPanelField....836
Establishing the appearance for the DynamicControlPanel....840
Container Tag and styling on DynamicControlPanel....841
Container Tag and styling on Fields....841
Additional Behavior Properties of the DynamicControlPanel....843

Adding a DynamicControlPanel Control....844

Properties of the DynamicControlPanel Control....847
Primary Properties....847
Container Appearance Properties....849
Behavior Properties....852

Properties on Field Classes....854

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

Properties common to all Field Classes 855
Container Properties 855
Appearance Properties 860
Behavior Properties 860
Event Handlers 861
Properties on NextPreviousPagerField, NextPagerField, and PreviousPagerField 862
Layout Pattern Properties 862
Buttons – Formatting Properties 864
Buttons – Content Properties 865
Buttons – Special Cases Properties 868
Properties on NumericPagerField 870
Layout Pattern Properties 870
Buttons – Formatting Properties 872
Sequential Numbering Properties 874
Buttons – Content Properties 875
Buttons – Special Cases Properties 878
Properties on all ExtendedNumericPagerFields 880
Extended Numbering Properties 881
Sequential Numbering Properties 882
Layout Pattern Properties 883
Buttons – Formatting Properties 885
Buttons – Content Properties 887
Buttons – Special Cases Properties 891
Properties on InfoPagerField 893
Layout Pattern Properties 893
Paging TextBox Properties 895
Properties on DropDownPageSizesField and LinksPageSizesField 897
Page Sizes Properties 897
Appearance Properties 898
Layout Pattern Properties 899
Properties on DropDownSorterField and LinksSorterField 900
SortKeys Properties 900
Appearance Properties 901
Layout Pattern Properties 902
AddSortKey method 903
Properties on CommandButtonField and NewCommandButtonField 904
Appearance Properties 904
Behavior Properties 906
Properties on DynamicButtonsField 907
Dynamic Buttons Properties 907
Properties on TemplateControlPanelField 908
Appearance Properties 908
Behavior Properties 910
Event Handlers 911

Constructors of the various Field Classes 913
[C#] 913
[VB] 915
NextPreviousPagerField 915
PreviousPagerField 915
NextPagerField 915
NumericPagerField 915
IncrementExtendedNumericPagerField 915
SpreadExtendedNumericPagerField 915
JumpByExtendedNumericPagerField 916
InfoPagerField 916
DropDownPageSizesField 916
LinksPageSizesField 916

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

DropDownSorterField 916
LinksSorterField 916
NewCommandButtonField 916
CommandButtonField 917
DynamicButtonsField 917
TemplateControlPanelField 917

FIELD TEMPLATES: THE USER INTERFACE FOR COLUMNS 918

Overview 919

Maintaining separation of concerns 920

Field Templates in the DES/FieldTemplates folder 921

Field Templates in the RadControls/FieldTemplates folder 924

Field Templates in the Ajax Controls Toolkit folder 925

CREATING AND MODIFYING FIELD TEMPLATES 926

Copy an Existing Template 926

Using one of DES's FieldTemplateUserControl classes 927
Steps for inheriting from a DES FieldTemplateUserControl class 929

SETTING PROPERTIES ON THE FIELD TEMPLATES 932

Choosing the right place to set properties on the Field Template 932

FIELD TEMPLATE CLASSES AND PROPERTIES 933
Properties Common to All Field Templates 934
Text.ascx Field Template 936
Properties 936
Supporting other controls to show the text 937
Text_Edit.ascx and MultilineText_Edit.ascx Field Templates 938
Properties 938
Integer.ascx, Decimal.ascx, Currency.ascx, Percent.ascx Field Templates 939
Properties 939
Integer_Edit.ascx, Decimal_Edit.ascx, Currency_Edit.ascx, Percent_Edit.ascx Field Templates 940
Properties 940
Measurement.ascx and Measurement_Edit.ascx Field Templates 941
Properties 941
All Date, DateTime, MonthYear, and Anniversary Field Templates 942
Properties 942
Time.ascx and Time_Edit.ascx Field Templates 943
Properties 943
Duration.ascx and Duration_Edit.ascx Field Templates 944
Properties 944
Enumerated.ascx Field Template 945
Properties 945
Enumerated_Edit.ascx Field Template 946
Properties 946
ForeignKey.ascx Field Template 947
Properties 947
ForeignKey_Edit.ascx Field Template 948

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

Properties 948
ManyToMany.ascx Field Template 949
Properties 949
ManyToMany_Edit.ascx Field Template 950
Properties 950
Boolean.ascx Field Template 951
Properties 951
BooleanLabel.ascx Field Template 952
Properties 952
BooleanCheckBox.ascx Field Template 953
Properties 953
Boolean_Edit.ascx Field Template 954
Properties 954
Children.ascx Field Template 955
Properties 955
DbImage.ascx and DbImage_Edit.ascx Field Templates 956
Properties on the Image 957
Properties for a ToolTip 958
Properties for a Label 958
Properties for an Expanded View 959
Properties for Editing the Image File 960
Creating Expanded View UserControls 962
Adding Virus Detection to uploaded files 964
UrlAsImage.ascx and UrlAsImage_Edit.ascx Field Templates 965
Properties on the Image 966
Properties for a ToolTip 967
Properties for a Label 967
Properties for an Expanded View 968
Properties for Editing the Url 968
Url.ascx Field Template 969
Properties 969
Url_Edit.ascx Field Template 970
Properties 970
EmailAddress.ascx Field Template 971
Properties 971
EmailAddress_Edit.ascx Field Template 972
Properties 972
Calendar_Edit.ascx Field Template 973
Properties 973

USING STYLE SHEETS WITHIN FIELD TEMPLATES 974

Adding Style Sheet Classes to your Field Templates 974

Changing the default path to the Field Templates Style Sheet File 974

CUSTOMIZATION BASED ON CATEGORIES 975
DESFieldTemplateUserControl.CheckCategories method 976

IMPLEMENTING THE ICUSTOMIZEFIELDTEMPLATE INTERFACE 977

Where to implement ICustomizeFieldTemplate 978

Adding the ICustomizeFieldTemplate Interface 979

CustomizeFieldTemplate method 980
Properties of the CustomizeFieldTemplateEventArgs Class 982

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

Methods of the CustomizeFieldTemplateEventArgs Class 984

FIELDTEMPLATEFACTORY: SELECTING THE CORRECT FIELD TEMPLATE FILE 988

FieldTemplates.config File Format 989
Aliases Section 990
<alias> Attributes 990
Pathsearch section 991
<path> Attributes 991
Templatesearch Section 992
<template> Attributes 992

Understanding how Field Templates are selected 993
Additional Rules 993

Creating Field Templates programmatically 994
Abandoning the Field Templates folder 994
How this works 994
Adjusting the list that maps UIHint to FieldTemplateUserControl type 995
Changing the controls output by a FieldTemplateUserControl 997
Using the CreateTemplate event handler 998

ASSISTANCE ON RELATED TECHNOLOGIES 1000

ASP.NET Dynamic Data 1000
Forums 1000
Relevant Blogs 1000
Articles 1000
Limiting Access to your PageTemplates 1000
Other 1000

LINQ to SQL 1001
Forums 1001
Blogs 1001
Articles 1001

ASP.NET Routing (System.Web.Routing namespace) 1002
Articles 1002

TROUBLESHOOTING 1003

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

License Information

This document includes information for features throughout the Peter's Data Entry Suite that are utilized with Microsoft's Dynamic Data for ASP.NET. You will need a license for the complete Suite or all of these modules: "Peter's Professional Validation", "Peter's More Validators", "Peter's TextBoxes", and "Peter's Date and Time". All other modules are optionally used as well.

In particular:

- DependencyAttribute uses the FieldStateController control in Peter's Interactive Pages.
- The Customizer has these properties that depend on Peter's Interactive Pages: DisableOnSubmit, ChangeMonitorEnables, DeleteConfirmMessage, UpdateConfirmMessage, CancelConfirmMessage, DependencyUI, and UseDependencyAttributes
- The FieldSecurityValidationAttribute requires Peter's Input Security.

Platform Support

Microsoft's Dynamic Data is first introduced in ASP.NET 3.5 Service Pack 1. While the rest of Peter's Data Entry Suite supports all versions of ASP.NET, this aspect is limited to ASP.NET 3.5 SP 1 and later, including ASP.NET 4. It includes a separate assembly compiled under .net 3.5 SP 1.

DES Dynamic Data either supports or mimics most Dynamic Data features introduced in ASP.NET 4.0, including attributes, Entity Templates, and FilterTemplates.

DES Dynamic Data also adapts ideas from Dynamic Data Futures (July 16, 2008 build). It has replacements for many concepts (including all Field Templates) and supports other concepts (including the DynamicObjectDataSource and most controls it offers.) It does not require the Microsoft.Web.DynamicData.dll, except if you need to use the features it supports.

Supports the DataSource controls from Versatile DataSources, a free product from PeterBlum.com.

Supports elements of the Ajax Control Toolkit, including some Field Templates. This support must be installed explicitly within the Web Application Updater. When installed, your web app's Dynamic Data folder will contain a Ajax Control Toolkit folder. If you must have the AjaxControlToolkit.dll assembly installed when this folder is installed.

Provides Field Templates that use controls from RadControls for ASP.NET AJAX (all versions). This support must be installed explicitly within the Web Application Updater. When installed, your web app's Dynamic Data folder will contain a RadControls folder. If you must have the Telerik_Web_UI.dll assembly installed when this folder is installed. See "Field Templates in the RadControls/FieldTemplates folder".

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

Technical Support and Other Assistance

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

Troubleshooting Section of this Guide

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

Developer's Kit

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

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

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

PeterBlum.Com MessageBoard

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

Getting Product Updates

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

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

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

Technical Support

ASP.NET Dynamic Data is a product of Microsoft. Microsoft provides technical support on it. PeterBlum.com provides technical support on its "DES Dynamic Data". There will be times when the two systems are difficult to distinguish. So please email PeterBlum.com with your question and if its more suited to Microsoft support, I will have you contact them. They have a fantastic free support forum at http://forums.asp.net/1145.aspx.

ASP.NET Dynamic Data uses several technologies extensively: ASP.NET Routing, LINQ to SQL, Entity Framework. The real concept of ASP.NET Dynamic Data itself is the DynamicField and DynamicControl, which deliver a user interface (Field Template) based on metadata describing a column. This User's Guide focuses on that and PeterBlum.com provides technical support for it. As for the other technologies, the section"Assistance on Related Technologies" should help you find the right resources. PeterBlum.com technical support does not provide support for them.

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

- Please review the Troubleshooting section first. See "Troubleshooting".
- See "Assistance on Related Technologies" when the issue involves ASP.NET Dynamic Data, LINQ to SQL, or ASP.NET Routing.
- 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.

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

- 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.

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

Overview of DES Dynamic Data

Click on any of these topics to jump to them:

- What is “Dynamic Data”?
- How Dynamic Data changes how you build web forms
- The Elements of Dynamic Data
  - Classes that Represent your Data
  - User Interface Elements
  - Attributes Assigned to the Classes Representing Your Database
  - Files and Folders
- How DES enhances ASP.NET Dynamic Data

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

What is "Dynamic Data"?

ASP.NET Dynamic Data is a technology from Microsoft that expands the toolset of ASP.NET. It is for building data entry web forms that interact with a database or classes that resemble a database. It was introduced in ASP.NET 3.5 Service Pack 1. So if you have an earlier version of ASP.NET, you will need to upgrade.

Dynamic Data was designed to solve a few specific problems:

- Provide a separation of concerns between business logic and web forms. Database-tier objects are used to describe the attributes of a column, such as validation rules and column name. The web form no longer needs to determine the right data entry control or validators. Instead, smart web controls gather and format those controls for you. They also supply two-way databinding.
- Provide a separation of concerns between the web form and the data oriented controls used to view and edit data. The web form declares a single control that identifies the column to show, and Dynamic Data swaps it for the correct controls to display or edit that column. For details, see "How Dynamic Data changes how you build web forms".
- Quickly create an application that can browse and edit your database by building "scaffolding" that reflects the columns on a table into controls like GridView and DetailsView.
- Quickly setup a very feature rich GridView, DetailsView, ListView, and FormView.

Note: ASP.NET Dynamic Data uses several technologies extensively: ASP.NET Routing, LINQ to SQL, Entity Framework. The real concept of ASP.NET Dynamic Data itself is in the DynamicControl, which deliver a user interface (Field Template) based on metadata describing a column. This User's Guide focuses on that and PeterBlum.com provides technical support for it. As for the other technologies, the section"Assistance on Related Technologies" should help you find the right resources. PeterBlum.com technical support does not provide support for them.

Advantages

From this, you should be thinking "I can just drop a single smart control on the page, assign its column name and get a very rich data entry control with validation and two way databinding!". Also consider these advantages:

- You can create a standard user interface for any type of data. For example, setup a DateTextBox as the editor for Dates. Dynamic Data will automatically use that DateTextBox user interface each time a Date is specified on the column. (This can be overridden.) In fact, you have create several different user interfaces for the same type of data. Perhaps you want a DateTextBox in some cases and a Calendar control in others.
- You are assured that the validators exactly match the business requirements. If those requirements change, attributes are modified on the database-tier objects and are automatically reflected on the web form.
- You can abandon using the BoundField element in a GridView or DetailsView. While it created a two-way databinding UI, you had no control over the features it offered, such as providing a DateTextBox for Date columns, or support of validators. Instead, you use Dynamic Data's smart "DynamicField" element to get a feature-rich user interface.
- Reduce or eliminate writing SQL code to interact with your tables. Most of this code is offloaded to specialized DataSource classes.
- Separates all data access code from your web forms so that modifications to your database and business logic do not require edits to your web forms.

Limitations

Dynamic Data cannot solve all data entry web form problems. In its first release, it has many limitations although by using DES, some of them have already been addressed. Here are the key limitations (taking into consideration DES's enhancements):

- Not every data entry scenario is covered. You need a DataSource object to interpret your data in a way that can be used by ASP.NET Dynamic Data. Microsoft provides a few database-tier objects: those generated by the LINQ to SQL and ADO.NET Entity Framework features found in ASP.NET 3.5.
    - PeterBlum.com has created Versatile DataSources, a free product containing two DataSource controls that support many new solutions, including allowing ADO.NET and POCO support.

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

- Your page must use a databinding oriented control, such as a GridView, DetailsView, ListView, FormView or third party grid, to hold Dynamic Data's smart DynamicField or DynamicControl.

DES Dynamic Data (this product) introduces many features to enhance ASP.NET Dynamic Data, overcoming its limitations. See "How DES enhances ASP.NET Dynamic Data".

Learn More

The official site: http://www.asp.net/dynamicdata/

A quick reference to the documentation: http://msdn.microsoft.com/en-us/library/cc488545.aspx

A quick reference to the API: http://msdn.microsoft.com/en-us/system.web.dynamicdata.aspx

Future developments for ASP.NET Dynamic Data:
http://www.codeplex.com/aspnet/Wiki/View.aspx?title=Dynamic%20Data&referringTitle=Home

See also "Assistance on Related Technologies".

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

How Dynamic Data changes how you build web forms

ASP.NET Dynamic Data emphasizes "separation of concerns" where business logic is separated from the web form. In addition, the web controls showing and validating data are shared by all web forms, providing a consistent user interface. It internally handles database interaction, so you don't have to write code to create, read, update, or delete records (often known as "CRUD").

In traditional ASP.NET web forms, the web form developer must setup every web control, including the right validators, leaving a potential of saving data that does not match your business logic. There is also the potential of creating a different look-and-feel on fields that are the same datatype. The database interaction is written into each form.

Separation of Concerns

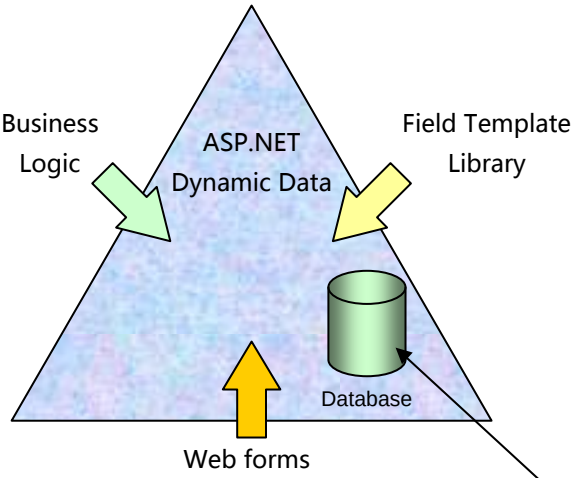
Here are the elements separated out by ASP.NET Dynamic Data:

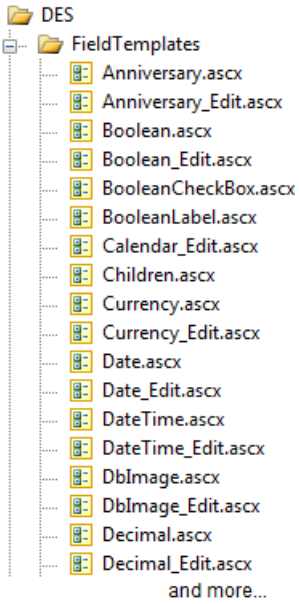
- Business logic – Rules for validation and other aspects of managing data on an individual column are kept within a file specific to your business logic. The web form developer never creates controls or code to handle this.
- Field Templates – User interface for displaying and editing the data types used by columns in your database. These are actually UserControls, subclassed from Dynamic Data's FieldTemplateUserControl class. Each Field Template handles a single data type, such as date and integer. Your web app will have a library of Field Templates. There can be several Field Template files for a single data type in your library, offering you different ways to display and edit that data type.
- Web Forms – The standard workspace for displaying HTML is still used to present the columns of your database. It still uses typical web controls, like Labels, Buttons, MultiViews, etc. When you need to show a column from your database, include a DynamicField or DynamicControl web control with the name of the column in its **DataField** property. ASP.NET Dynamic Data internally converts that web control into the right Field Template and applies the business logic to it. Alternatively, let the automatic scaffolding feature populate a Page Template with a list of columns.
- Code to create, read, update and delete records ("CRUD") is handled for you. ASP.NET Dynamic Data relies on special DataSource controls, such as LinqDataSource or EntityDataSource, that are placed on the Web Forms.

```
[MetadataType(
   typeof(CategoryMetadata))]
public partial class Category
{
}

[DisplayName("Categories")]
public class CategoryMetadata
{
  [DES.Required()]
  [DES.DisplayName("Name")]
  public object CategoryName
  { get; set; }

  [DES.DataType(
    DataType.MultilineText)]
  [DES.StringLength(2000)]
  public object Description
  { get; set; }
}
```






```
<asp:DynamicField DataField="LastName" />
<asp:TemplateField>
  <asp:DynamicControl DataField="FirstName">
  <asp:DynamicControl DataField="MiddleInitial">
</asp:TemplateField>
<asp:DynamicField DataField="BirthDate"
       UIHint="Calendar" />
```

DataSource handles CRUD

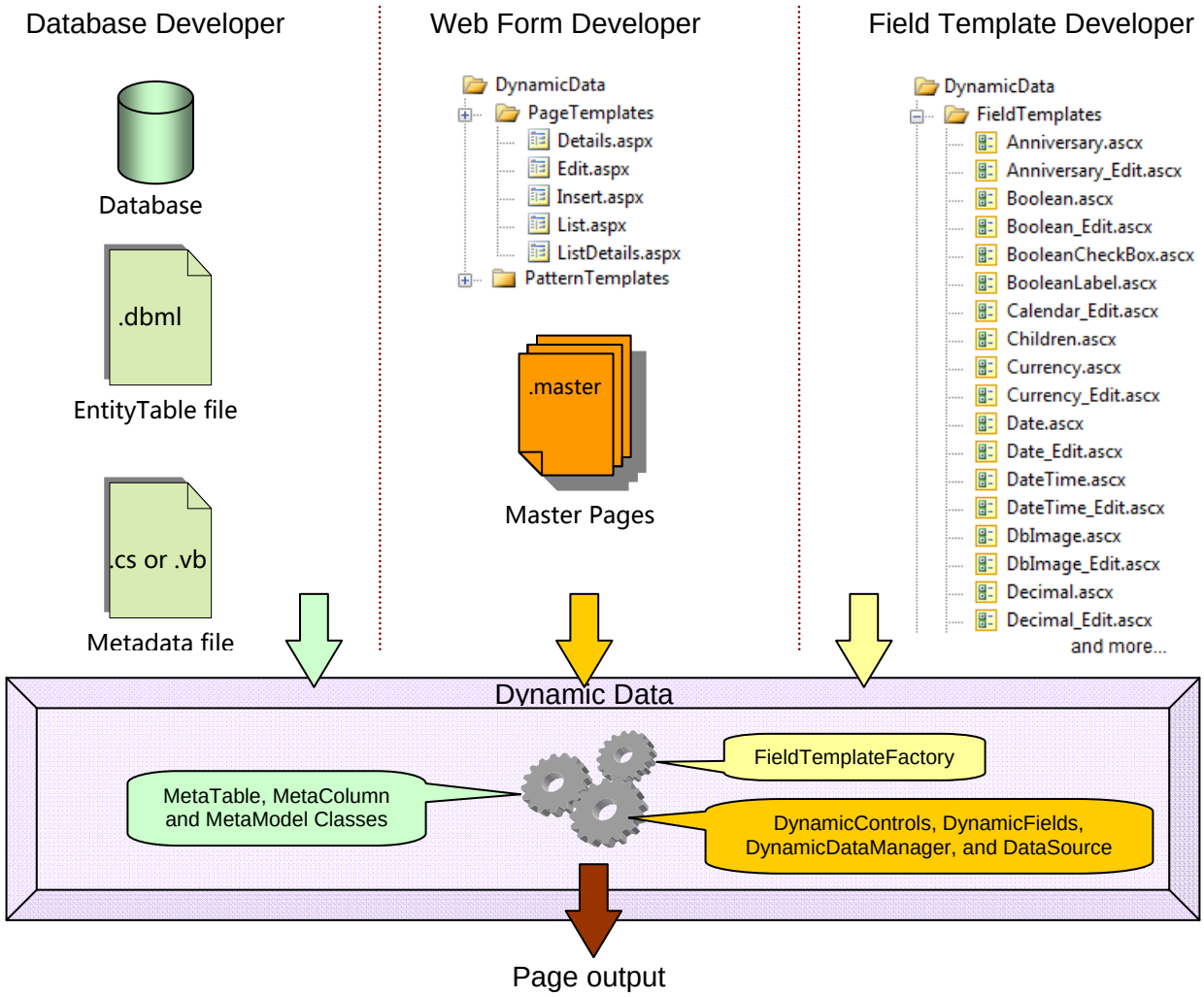
Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Developers can play Different Roles

When creating a Dynamic Data-based application, you have several developer roles to fill:

- The database developer, who creates the EntityTable and MetaData Classes. He assign attributes and code to the EntityTable MetaData Classes that dictate the business logic of each column, including validation rules.
- The Field Template developer, who builds a smart user interface specific to a single type of data, including readonly and editable modes, complete with validators.
- The web form developer defines the columns and their layout of the page, either through scaffolding and Page Templates, or by DataBound controls (like ListView and FormsView) hosting DynamicControls and DynamicPatterns.

Dynamic Data combines these elements through the following controls found on the web forms: DynamicControls, DynamicFields, DynamicDataManager and your DataSource.



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

Structure of a Dynamic Data driven web form

Every Dynamic Data web form has 3 primary controls:

- DataBound control – Container for the data oriented areas of your web form, providing read-only and editable ways to interact with properties of an EntityTable object supplied to it. Examples include GridView, DetailsView, ListView, and FormView.

  You add DynamicControl or DynamicField objects to specify which property on EntityTable is shown by using a Field Template. Field Templates provide DataBinding between the EntityTable object and the controls they host.

- DataSource control – Used by DataBound controls to get and set the object containing data. Examples include LinqDataSource (supports LINQ to SQL) and EntityDataSource (supports ADO.NET Entity Framework.)

  Dynamic Data requires that the DataSource control implements the System.Web.DynamicData.IDynamicDataSource interface, which most DataSources do *not*. Choose from LinqDataSource, EntityDataSource, DomainDataSource, and those found in the Versatile DataSources project.

- DynamicDataManager control – Connects Dynamic Data to your DataBound control and establishes some of the behaviors that you want Dynamic Data to add to the DataBound control.

```
<desDD:DynamicDataManager ID="DDM" runat="server" >
  <Adapters>
    <desDD:SomeDataControlAdapter DataBoundControlID="DBView1" />
  </Adapters>
</desDD:DynamicDataManager>

<asp:someView ID="DBView1" runat="server" DataSourceID="DataSource1">
DynamicControls, web controls, and HTML to format this content
</asp:someView>

<asp:someDataSource ID="DataSource1" runat="server">
</asp:someDataSource>
```

Most of your work is inside the DataBound control. For example, with the FormView control, this provides editing features for the Category table of the NorthWind database.

```
<asp:FormView ID="DBView1" runat="server" DataSourceID="DataSource1">
<ItemTemplate>
   Category Name:
   <desDD:DynamicControl ID="Name" runat="server" DataField="Name" />
   Description:
   <desDD:DynamicControl ID="Description" runat="server" DataField="Description" />
   Picture:
   <desDD:DynamicControl ID="Picture" runat="server" DataField="Picture" />
</ItemTemplate>
</asp:FormView>
```

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

When in Edit or Insert modes

If your web form offers editing capabilities, also add the ValidationSummary to the page to report errors. Some errors are not field specific and will only appear in the ValidationSummary control.

```
<desDD:DynamicDataManager ID="DDM" runat="server" >
  <Adapters>
    <desDD:SomeDataControlAdapter DataBoundControlID="DBView1"
         ValidationSummaryControlID="ValidationSummary1"
         SupportsEditMode="true" SupportsInsertMode="true" />
  </Adapters>
</desDD:DynamicDataManager>

<des:ValidationSummary ID="ValidationSummary1" runat="server" />

<asp:someView ID="DBView1" runat="server" DataSourceID="DataSource1">
  DynamicControls, web controls, and HTML to format this content
</asp:someView>

<asp:someDataSource ID="DataSource1" runat="server">
</asp:someDataSource>
```

The DynamicControls must have their **Mode** property set to `Edit` or `Insert` to use the Field Template associated with edit or insert mode.

```
<asp:FormView ID="DBView1" runat="server" DataSourceID="DataSource1">
<EditItemTemplate>
   Category Name:
   <desDD:DynamicControl ID="Name" runat="server"
       DataField="Name" Mode="Edit" />
   Description:
   <desDD:DynamicControl ID="Description" runat="server"
       DataField="Description" Mode="Edit" />
   Picture:
   <desDD:DynamicControl ID="Picture" runat="server"
       DataField="Picture" Mode="Edit" />
   <des:Button ID="Submit" runat="server" Text="Save" CommandName="Update" />
</EditItemTemplate>
</asp:FormView>
```

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

The Elements of Dynamic Data

ASP.NET Dynamic Data introduces a number of classes, controls and attributes. Here is a brief overview of them.

Click on any of these topics to jump to them:

- Classes that Represent your Data
- User Interface Elements
- Attributes Assigned to the Classes Representing Your Database
- Files and Folders

Classes that Represent your Data

Your data is represented by a class. There are two models of data: CRUD ("Create, read, update, and delete" which is associated with databases) and POCO ("Plain old CLR Object" which is associated with non-CRUD situations). CRUD is the way Microsoft has represented its examples, but POCO is supported as well.

Data Context – A class that describes the overall database in the database tier. The class has properties that declare its tables. It is created by LINQ to SQL or ADO.NET Entity Framework. The class name is assigned to the **ContextDataType** property of your LinqDataSource or EntityDataSource.

For examples based on the NorthWind database and LINQ to SQL:

[C#]

[DES Product Folder]/DynamicData/CS/Samples/App_Code/NorthWind.designer.cs

[VB]

[DES Product Folder]/DynamicData/VB/Samples/App_Code/NorthWind.designer.vb

EntityTable – Classes defined by LINQ to SQL and the ADO.NET Entity Framework that represent tables in the database tier. These classes have the same name as the corresponding table in your database. Properties are the type and name of the corresponding column in the table. See also EntityTable Metadata Class below.

For examples based on the NorthWind database and LINQ to SQL:

[C#]

[DES Product Folder]/DynamicData/CS/Samples/App_Code/NorthWind.designer.cs

[VB]

[DES Product Folder]/DynamicData/VB/Samples/App_Code/NorthWind.designer.vb

EntityTable MetaData Class – Dynamic Data uses metadata to annotate the business logic of tables. Metadata is usually Attributes, assigned to the EntityTable's properties. Because LINQ to SQL and ADO.NET Entity Framework will be asked to occassionally regenerate these classes, you **do not** edit the EntityTable classes directly. Instead, you will create an extra class that parallels the EntityTable class where metadata is applied. This class is called the "EntityTable MetaData Class".

For examples based on the NorthWind database and LINQ to SQL:

[C#]

[DES Product Folder]/DynamicData/CS/Samples/App_Code/NorthWindMetadata.cs

[VB]

[DES Product Folder]/DynamicData/VB/Samples/App_Code/NorthWindMetadata.vb

POCO Type – Same idea as the EntityTable, but for non-CRUD situations. Requires the POCODataSource control of Versatile DataSources (free on CodePlex).

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

MetaColumn class – This class knows about a single column, such as its name and type. It keeps a list of attributes that you defined on the EntityTable MetaData Class in its **Attributes** property. It exposes the value of many of them in other properties, like **DisplayName** and **ColumnType** (the datatype of the property).

MetaTable class – Dynamic Data automatically creates one of these for each EntityTable in your Data Context. It gathers all of the knowledge about this table's structure and business logic. A key property is **Columns**, which is a collection of MetaColumns.

MetaModel – Defines a list of MetaTables that will be utilized by your Dynamic Data application. Always create at least one of these in the `Application_Start()` method of your web application by calling the `RegisterContext()` method. If you have several database schemas, you may have multiple MetaModels.

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

User Interface Elements

Click on any of these topics to jump to them:

- Working with Data Fields
- Working with DataBound controls
- Controls used within DataBound controls and PatternTemplates
- Other Controls

Working with Data Fields

Field Template file – Defines the user interface for a specific datatype. There are many of them, not just for the many datatypes, but also for their usage: non-editable, edit mode, and insert mode. Their file names tell you exactly what they are for by including the data type and if in edit or insert mode, "_Edit" or "_Insert". For example, "Date.ascx", "Date_Edit.ascx" and "Date_Insert.ascx".

You don't have to create Field Templates for everything. Dynamic Data knows how to fallback to a similar Field Template when it can't find the exact one. As a result, when you look at the list of Field Templates that come with Dynamic Data, you will not see any for Insert mode. It knows to fallback to those for Edit mode. See "Field Templates: The user interface for columns".

Explore the Field Templates shipped with DES here:

[C#]

[DES Product Folder]/DynamicData/CS/FieldTemplates

[VB]

[DES Product Folder]/DynamicData/VB/FieldTemplates

FieldTemplateUserControl class – The base class for all Field Templates. It is actually subclassed from System.Web.UI.UserControl. Each FieldTemplateUserControl references a single MetaColumn in its **Column** property. It prepares the data entry control and its validators according to the attributes of the MetaColumn.

DESFieldTemplateUserControl class – *DES specific.* The base class for all Field Templates shipped with Peter's Data Entry Suite. It is subclassed from FieldTemplateUserControl and introduces numerous extensions, including support for the DES Validation Framework. See "Using one of DES's FieldTemplateUserControl".

FieldTemplateFactory – A class used to convert the MetaColumn's information into an actual Field Template control object. The native one is automatically setup for you.

DESFieldTemplateFactory – *DES specific*. A subclass of FieldTemplateFactory that greatly expands the capabilities of the FieldTemplateFactory. You define special cases such as aliases and alternative locations in the **FieldTemplates.config** file. This FieldTemplateFactory follows those special cases. An alias allows you to globally map the Field Template name requested for a specific datatype to another, such as always use **Calendar_Edit.ascx** instead of **Date_Edit.ascx** on a Date column. An alternative location allows you to have several cases of the same named Field Template in different folders and map which will be actually loaded. See "FieldTemplateFactory: Selecting the correct Field Template file".

For an example:

[C#]

[DES Product Folder]/DynamicData/CS/Samples/App_Code/DynamicData.cs

[VB]

[DES Product Folder]/DynamicData/VB/Samples/App_Code/DynamicData.vb

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

Working with DataBound controls

DataBound controls – The controls used to show lists and single record views of data. They are generally subclassed from `System.Web.UI.WebControls.BaseDataBoundControl` and include: GridView, DetailsView, ListView, FormView, DynamicListView, and DynamicFormView.

Page Template file – A web form that includes the primary controls for Dynamic Data: a DataSource, DataBound Control, and DynamicDataManager to display the contents of a table. They are located in the **[web app]/DynamicData/PageTemplates** folder. Each file maps to an action: Details, Edit, Insert, List, and List+Details. They are table and data field agnostic. They acquire the table from ASP.NET URL Routing (such as /Products/Edit.aspx). They acquire the data fields from automatic scaffolding. As a result, they are a specialized way to use Dynamic Data within your web forms. With Page Templates, you can create a fully operational database exploration application representing all tables and columns that are declared in your Data Context.

See Building Applications with PageTemplates" and "Building Applications without PageTemplates".

Explore the Page Templates shipped with DES here:

[C#]

[DES Product Folder]/DynamicData/CS/PageTemplates

[VB]

[DES Product Folder]/DynamicData/VB/PageTemplates

Automatic Scaffolding – Defines the data fields shown tables in DataBound controls so you don't have define them manually. For example, the GridView control will populate its **Columns** property with DynamicFields when you set **AutoGenerateColumns** to `true`. Similarly, the DynamicListView and DynamicFormView controls will populate its **ItemsInPattern** property with DataFieldInPattern objects when its **ItemsInPattern** collection is empty or you set its **AutoScaffoldDataFields** property to `True`.

Page Templates are setup with this feature enabled, allow them to work with any table.

Automatic Scaffolding uses the EntityTable class to determine the list of columns and their order. It uses the ScaffoldColumnAttribute to hide or move columns.

DES Dynamic Data extends the native scaffolding with a more flexible FieldGenerator (see below), enhanced ScaffoldColumnAttribute, and ways for you to programmatically control the scaffolding process. See "Automatic Scaffolding Overview".

FieldGenerator – A class implementing System.Web.UI.IAutoFieldGenerator that performs Automatic Scaffolding (see above). This class creates a list of field objects to display. The type of object depends on the DataBound control. For GridView and DetailsView, it generates DynamicFields. For DynamicListView, DynamicFormView, and DynamicPattern, it generates DataFieldInPattern objects. See "Customizing Automatic Scaffolding".

PatternTemplate – A special UserControl that defines a pattern of controls and HTML, including DynamicControls (DynamicData's control that converts a column's name into a Field Template). Web page development frequently uses patterns, such as the HTML for a table row with two columns, where the first column contains a field name and the other is the field's data.

Patterns can be very complex. A GridView control is really a pattern, with a table tag enclosing numerous table rows of data. DES includes PatternTemplates that mimic the GridView and DetailsView. PatternTemplates have a significant advantage over these controls: you have full control over the HTML they generate. As a result, you will likely use the DynamicListView and DynamicFormView controls instead of GridView and DetailsView, because of this flexibility. DynamicListView and DynamicformView are introduced in DES Dynamic Data and are built around PatternTemplates.

> *Note: An upcoming release of ASP.NET Dynamic Data introduces "Entity Templates". That is a very simplistic implementation of PatternTemplates. While you can use Entity Templates within DES Dynamic Data, you will find PatternTemplates a better toolset.*

PatternTemplate files are stored in **[web app]/DynamicData/PatternTemplates**.

See "Pattern Templates: Reusable formatting for DynamicControls".

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

For examples using DynamicListView and DynamicFormView:

[C#]

[DES Product Folder]/DynamicData/CS/Samples/Using PatternTemplates

[VB]

[DES Product Folder]/DynamicData/VB/Samples/Using PatternTemplates

Named Styles – Use by PatternTemplates to allow you to override its default formatting. You create NamedStyle objects, with style information and a name known to the PatternTemplate file when use a DynamicPattern, DynamicListView, or DynamicFormView control. See "Named Styles: Your pages control the formatting".

ItemInPattern classes – Objects subclassed from `PeterBlum.DES.DynamicData.BaseItemInPattern` that are used to describe the data fields consumed by a PatternTemplate file. You add these objects to the ItemsInPattern collection of DynamicPattern, Fields collection of DynamicFormView, and Columns collection of DynamicListView. The subclasses are:

DataFieldInPattern class – Assigns a data field name to the DynamicControl of an ItemTemplate. See "The DataFieldInPattern Class".

DataFieldListInPattern class – Assigns a list data field names to the DynamicControls of an ItemTemplate that has multiple DynamicControls. See "The DataFieldListInPattern Class".

ScaffoldInPattern class – Create a list of DataFieldInPattern objects using Automatic Scaffolding. Each uses a separate ItemTemplate instance for its data field name. See "The ScaffoldInPattern Class".

ButtonsInPattern – Uses a NamedPart to provide buttons. It can provide titles to the DynamicColumnTitle and DynamicLabel controls in the header and footer rows through its **HeaderText**, and **FooterText** properties. See "The ButtonsInPattern Class".

LabelInPattern – Uses a NamedPart to assign text to a DynamicLabel control through its **LabelText**, **HeaderText**, and **FooterText** properties. See "The LabelInPattern Class".

NoDataFieldsInPattern class – Uses a NamedPart to provide additional formatting that lacks DynamicControls and DynamicLabels. See "The NoDataFieldsInPattern Class".

TemplateNotInPattern class – Declares HTML and web controls, including DynamicControls and DynamicLabels, without having to modify the PatternTemplate. See "The TemplateNotInPattern Class".

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

Controls used within DataBound controls and PatternTemplates

DynamicListView control – A subclass of the ListView control that can be used in place of GridView or ListView. It uses PatternTemplates to deliver the quick setup of GridView and flexible formatting of ListView. See "The DynamicListView Control".

DynamicFormView control – A subclass of the FormView control that can be used in place of the DetailsView or FormView. It uses PatternTemplates to deliver the quick setup of DetailsView and flexible formatting of FormView. See "The DynamicFormView Control".

DynamicControl control – A web control that uses the MetaTable and MetaColumn to choose a Field Template. Drop one on the page, give it a data field name in the **DataField** property and it gives you back a databound UI for displaying or editing that data. If you want different user interface, create a Field Template for it and assign the Field Template's name to the **UIHint** property. This control is used in the Template properties of all DataBound controls. See "DES.DynamicControl: The heart of Dynamic Data".

DynamicField class – Used on GridView and DetailsView as a replacement for the BoundField. Its underlying technology is really just the DynamicControl with a wrapper that makes it operate within the GridView and DetailsView. When you use Dynamic Data's scaffolding feature, it will create these controls for you. See "DES.DynamicField: Using Dynamic Data in GridView and DetailsView".

DynamicPattern control – Inserts a PatternTemplate into the DataBound control and assigns the data field names to the DynamicControls contained within the PatternTemplate. See "The DynamicPattern Control".

DynamicLabel control – A Label control that supports assigning a DynamicControl to its **AssociatedControlID** property. (Internally, it alters the AssociatedControlID to use the actual control that displays/edits the data.) It also get defined its own Text property based on the colum name specified. See "The DynamicLabel Control".

DynamicButtons control – Similar to the CommandField object used by GridView and DetailsView, this control generates the correct buttons based on the data entry mode. For ReadOnly, the buttons can be: New, Edit, Delete, Details, and Select. For Edit and Insert, the buttons can be Save and Cancel. These buttons direct the DataBound control to take a specific action. See "The DynamicButtons Control".

DynamicColumnTitle control – Displays the name of a data field and optionally provides it with a sorting user interface (using the DataBound controls to implement the actual sorting). Use it in the header of a list of records (like ListView). See "The DynamicColumnTitle Control".

DynamicHyperLink control – Requires Dynamic Data Futures or later build. A control that generates links to table actions based on the current routing rules. It can also data bind to table rows, allowing to display links specific to the current item.

DynamicTagStyles control – Used within PatternTemplates to create HTML tags that have styles and other attributes from the Named Styles supplied by the DynamicPattern, DynamicListView, or DynamicFormView controls. See "The DynamicStylesTag Control".

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

Other Controls

DynamicDataManager – A control that sets up the DataBound controls (GridView, DetailsView, etc) with the right table. For controls that support DynamicFields, it generates the list of columns when using the Scaffolding feature. DES provides a subclass of the one shipped in Dynamic Data to simplify a few things. See "DynamicDataManager Control and Adapters".

For an example using DES's DynamicDataManager:

[C#]

[DES Product Folder]/DynamicData/CS/Samples/DetailsViewOnProducts.aspx

[VB]

[DES Product Folder]/DynamicData/VB/Samples/DetailsViewOnProducts.aspx

DataBound control adapter – *DES specific*. Connects the DataBound controls to Dynamic Data. Adapters do a lot of work in setting up the DataBound control, offloading tasks that you find embedded into native Page Templates. By removing the code from a Page Template, it reduces the chance of bugs as you edit and clone Page Templates. It also introduces a number of DES-specific tasks, like setting up buttons to disable on submit.

DataBound Control Adapters are declared in the **Adapters** property of the DynamicDataManager control. There are separate DataBound Control Adapter classes for each type of DataBound control, such as GridViewAdapter and FormViewAdapter. See "DynamicDataManager Control and Adapters".

DynamicValidator – Not every validation rule can be expressed at the user interface level in a Validator control. Often the database developer will enhance their EntityTable MetaData Classes with validation logic. If that logic detects an error, it throws an exception (specifically `System.ComponentModel.DataAnnotations.ValidationException`). The DynamicValidator captures that exception and displays it's error message.

DynamicValidator is actually a real server-side only validator that sets **IsValid** to `false` when it catches an exception. DES enhances this by automatically installing these validators. You usually don't have to add these to your web forms. See "DynamicValidator Control".

For an example using the DES DynamicValidator:

[C#]

[DES Product Folder]/DynamicData/CS/Samples/Using DataBoundControlAdapters/FormView.aspx

[VB]

[DES Product Folder]/DynamicData/VB/Samples/Using DataBoundControlAdapters/FormView.aspx

Customizer – *DES specific*. This web control interacts with the DES-supplied Field Templates, Page Templates, and PatternTemplates, customizing their behavior. This avoids having you edit the actual Field- and Page Templates in many cases. For example, if you want the **Currency_Edit.ascx** Field Template to show the currency symbol in its CurrencyTextBox, the Customizer offers the property **CurrencyTextBox_UseCurrencySymbol**. The Customizer's **ErrorMessages** property is one of its most valuable as it is where you can define the error messages for your validators, again without editing the actual Field Templates. See "Customizer Control".

For an example using the Customizer on a master page for your Page Templates:

[C#]

[DES Product Folder]/DynamicData/CS/Samples/Site.master

[VB]

[DES Product Folder]/DynamicData/VB/Samples/Site.master

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

Attributes Assigned to the Classes Representing Your Database

The following attributes are assigned to properties on the EntityTable MetaData Class to define all kinds of rules. The MetaColumn reads these attributes and sets itself up from them.

Reminder: An Attribute is a class derived from System.Attribute that is assigned to code elements like methods, classes and properties. In C#, they are enclosed in [] characters while in VB, they are enclosed in <> characters.

Examples of many of the following Attributes are found:

[C#]

[DES Product Folder]/DynamicData/CS/Samples/App_Code/NorthWindMetadata.cs

[VB]

[DES Product Folder]/DynamicData/VB/Samples/App_Code/NorthWindMetadata.cs

ValidationAttribute - A base class for validation rules. It supplies the ErrorMessage that will be transferred to the associated validator. (Of course, ErrorMessage is a UI concept so this is merely a suggestion and is there when the page developer doesn't set one up elsewhere. If you don't supply an ErrorMessage, ValidationAttribute will be retrieved from the Customizer or use a default.) Subclasses supply specific validation rules. See "ValidationAttributes: Applying Constraints on Columns".

RequiredAttribute, **RangeAttribute**, **StringLengthAttribute**, **RegularExpressionAttribute** - The ValidationAttribute subclasses supplied with ASP.NET Dynamic Data. Normally you will use their DES equivalents listed below.

BaseDESValidationAttribute – *DES specific.* A base class for validation rules that create DES framework validators. It is subclassed from ValidationAttribute (above). Its properties support these properties on DES validators: ErrorMessage, ErrorMessageLookupID, SummaryErrorMessage, and SummaryErrorMessageLookupID. See "ValidationAttributes: Applying Constraints on Columns".

DES includes the following:

- **RequiredAttribute** – Creates the RequiredTextValidator or RequiredSelectionValidator, depending on the type of data entry control. Has parameters for UnassignedValues and UnassignedCaseInsensitive, corresponding to properties on the RequiredTextValidator. ASP.NET Dynamic Data also has a RequiredAttribute and its supported by DES Dynamic Data, but its not as flexible.
- **RangeAttribute** – Creates the RangeValidator. Has parameters for Minimum and Maximum. They must conform to the datatype of the column to which this attribute is assigned. ASP.NET Dynamic Data also has a RangeAttribute and its supported by DES Dynamic Data, but its not as flexible.
- **CompareToValueAttribute** – Creates the CompareToValueValidator. Has parameters for Operator and ValueToCompare (which must conform to the datatype of the column to which this attribute is assigned), corresponding to properties on the validator.
- **RegularExpressionAttribute** – Creates the RegExValidator. Has parameters for Expression, CaseInsensitive, Multiline and IgnoreBlankText, corresponding to properties on the validator. DES also stores a list of expressions which you can use by supplying the name of the expression in the NameOfExpression parameter. ASP.NET Dynamic Data also has a RegularExpressionAttribute and its supported by DES Dynamic Data, but its not as flexible.
- **CharacterSetAttribute** – Creates a CharacterValidator and makes the Field Templates that use a FilteredTextBox use the defined character set. Those templates include Text_Edit.ascx, Multiline_Edit.ascx, and anything subclassed from `PeterBlum.DES.DynamicData.BaseTextBoxFTUC`. Has parameters for LettersUppercase, LettersLowercase, Digits, Space, Enter, DiacriticCharacters, Punctuation, VariousSymbols, MathSymbols, CurrencySymbols, and EnclosureSymbols, corresponding to properties on the validator and FilteredTextBox.
- **StringLengthAttribute** – Creates the TextLengthValidator. Has parameters for Minimum and Maximum, corresponding to properties on the validator. ASP.NET Dynamic Data also has a StringLengthAttribute and its supported by DES Dynamic Data, but its not as flexible.
- **CompareTwoColumnsAttribute** – Creates the CompareTwoFieldsValidator. Has parameters for Operator and SecondColumnName, where you give the name of the column being compared.
- **DifferenceAttribute** - Creates the DifferenceValidator. Has parameters for DifferenceValue, Operator and SecondColumnName, where you give the name of the column being compared.

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

- **StringListAttribute** – Creates the CompareToStringsValidator. Supply a delimited list of strings that are required in a string-type column.
- **InjectionSecurityAttribute** – Creates the FieldSecurityValidator to override page-level security rules for SQL Injection and Cross Site scripting attacks. Has parameters for DetectSQLInjection, SQLInjectionLevel, DetectScriptInjection, HTMLTagMode, and HTMLTag, corresponding to properties on the validator.
- **CreditCardNumberAttribute** – Creates the CreditCardNumberValidator. Has parameters for IgnoreBlankText, CheckByBrand, and AllowSpaces, corresponding to properties on the validator.
- **ABARoutingNumberAttribute** – Creates the ABARoutingNumberValidator.

DataTypeAttribute – Each property has a type, such as string or int, that mimics the type from the Column in the table (nvarchar, integer). That type is not specific enough for developing a user interface or validation rules. For example, a double can be used as a currency, percentage, duration, longitude, etc. DateTime can be used as a Date, Time, both or Duration. The DataTypeAttribute lets you refine the type so the right Field Template can be selected. Instead of just getting a textbox that validates decimals, it may get a CurrencyTextBox or PercentageTextBox with right validators.

There are two DataTypeAttribute classes. `System.ComponentModel.DataAnnotations.DataTypeAttribute` is the original. Its parameter takes a value from the enumerated type `System.ComponentModel.DataAnnotations.DataType`. DES has created several more types than found in `System.ComponentModel.DataAnnotations.DataType`. So it introduces `PeterBlum.DES.DynamicData.DataTypeAttribute` which takes the same parameters as the original, plus supports another enumerated type, `PeterBlum.DES.DynamicData.ExtendedDataType`, with its additional types.

DataTypeAttribute is actually a subclass of ValidationAttribute, giving it validation capabilities for a specific datatype. DES supplies numerous subclasses to handle all kinds of datatypes. They are:

- **IntegerDataTypeAttribute** – Declares the column is an integer and supplies a DataTypeCheckValidator for Integer. Supports several integer types. When assigned, it selects these Field Templates: Integer.ascx, Integer_Edit.ascx, and Integer_Insert.ascx.
- **DecimalDataTypeAttribute** – Declares the column is a decimal and supplies a DataTypeCheckValidator for Decimal. Supports decimal and double types. When assigned, it selects these Field Templates: Decimal.ascx, Decimal_Edit.ascx, and Decimal_Insert.ascx.
- **CurrencyDataTypeAttribute** – Declares the column is a currency and supplies a DataTypeCheckValidator for Currency. Supports decimal and double types. When assigned, it selects these Field Templates: Currency.ascx, Currency_Edit.ascx, and Currency_Insert.ascx.
- **PercentDataTypeAttribute** – Declares the column is a decimal and supplies a DataTypeCheckValidator for Percent. Supports decimal and double types. When assigned, it selects these Field Templates: Percent.ascx, Percent_Edit.ascx, and Percent_Insert.ascx.
- **IntegerMeasurementDataTypeAttribute** – A measurement is a numeric column that reflects a measurement, like gallon, millimeter, and degrees. This type lets you define labels that describe the measurement and can appear on the web form. Declares the column is a integer and supplies a DataTypeCheckValidator for Integer. Supports several integer types. When assigned, it selects these Field Templates: Measurement.ascx, Measurement_Edit.ascx, and Measurement_Insert.ascx.
- **DecimalMeasurementDataTypeAttribute** – Same as IntegerMeasurementDataTypeAttribute, but supports decimal and double types.
- **DateDataTypeAttribute** – Declares the column is a Date and supplies a DataTypeCheckValidator for Dates. Supports DateTime types. When assigned, it selects these Field Templates: Date.ascx, Date_Edit.ascx, and Date_Insert.ascx.
- **TimeOfDayDataTypeAttribute** – Declares the column contains the time of day (time from 0 to 23:59:59) and supplies a DataTypeCheckValidator for TimeOfDay. Supports DateTime types. When assigned, it selects these Field Templates: Time.ascx, Time_Edit.ascx, and Time_Insert.ascx.
- **DurationDataTypeAttribute** – Declares the column contains the duration and supplies a DataTypeCheckValidator for Duration. Supports DateTime, integer, and decimal types. When using integer or decimal, use the Conversion

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

parameter to indicate if the value is in milliseconds, seconds, minutes, hours, or days. When assigned, it selects these Field Templates: Duration.ascx, Duration_Edit.ascx, and Duration_Insert.ascx.

- **AnniversaryDataTypeAttribute** – Declares the column is an anniversary (date without the year) and supplies a DataTypeCheckValidator for Anniversary. Supports DateTime types. When assigned, it selects these Field Templates: Anniversary.ascx, Anniversary_Edit.ascx, and Anniversary_Insert.ascx.
- **MonthYearDataTypeAttribute** – Declares the column is a MonthYear (date without the day of the month) and supplies a DataTypeCheckValidator for MonthYear. Supports DateTime types. When assigned, it selects these Field Templates: MonthYear.ascx, MonthYear_Edit.ascx, and MonthYear_Insert.ascx.
- **EnumeratedAttribute** – For integer columns that have a fixed list of values and associated text for each value, much like an enumerated type. In fact, you can pass an enumerated type to this attribute and it will learn the correct mapping of number to text. You can also create other mappings and support localization. Field Templates generally show a list or DropDownList with the textual values. When assigned, it selects these Field Templates: Enumerated.ascx, Enumerated_Edit.ascx, and Enumerated_Insert.ascx.
- **EnumeratedStringAttribute** – Same as EnumeratedAttribute, but for string type columns. Lets you limit the strings offered in a field using a list. You can supply alternative text and localized text.
- **DbImageDataTypeAttribute** – For binary columns that contain an image, including gif, jpg, png, tiff, and bmp. When assigned, it selects these Field Templates: DbImage.ascx, DbImage_Edit.ascx, and DbImage_Insert.ascx.
- **UrlAsImageDataTypeAttribute** – For string columns containing a Url to an image. Its Field Templates can display the URL and/or the image. When assigned, it selects these Field Templates: UrlAsImage.ascx, UrlAsImage_Edit.ascx, and UrlAsImage_Insert.ascx.
- **EmailAddressDataTypeAttribute** – For Email addresses. Creates the EmailAddressValidator. Has parameters for MultipleAddressesAllowed, DelimiterCharacter, and DelimiterAllowsSpaces, corresponding to properties on the validator. When assigned, it selects these Field Templates: EmailAddress.ascx, EmailAddress_Edit.ascx, and EmailAddress_Insert.ascx.
- **UrlDataTypeAttribute** – For Urls. Creates the RegexValidator to evaluate the Url. When assigned, it selects these Field Templates: Url.ascx, Url_Edit.ascx, and Url_Insert.ascx.

DependencyAttribute – When the column does not need data unless another column has a specific value, this attribute can disable validators or the data entry control on the column. See "DependencyAttributes: Relationships between columns".

UIHintAttribute – Overrides the default Field Template so you can use different user interfaces for a specific datatype. For example, you may want to use a Calendar instead of a DateTextBox to select a date. Just specify the template name, without "_Edit", "_Insert", or ".ascx". Both DynamicControl and DynamicField controls have the equivalent functionality in their **UIHint** property. See "UIHintAttribute: Customizing the Field Template used".

DisplayNameAttribute – Gives a user friendly label for the column name. That label will appear:

- A column title of the GridView control
- The row label of the DetailsView control.
- PatternTemplates will assign it to any DynamicLabel control that is associated with a DynamicControl.
- In the DynamicColumnTitle control.
- In the Filter control.

See "DisplayNameAttribute: Friendly column name".

DescriptionAttribute – Descriptive text that the Field Template displays as a hint (usually a tooltip although with DES, it can be a PopupView). See "DescriptionAttribute: A Hint for the user".

DefaultValueAttribute – Provides a default value for columns when the user starts a new row in the table. See "DefaultValueAttribute: Initial value when creating a new record".

DisplayFormatAttribute – Assists the Field Template when it formats a value as a string. Provides these properties that you already find in a BoundField: **NullDisplayText** (used when a column is null and needs to be displayed in a label), **DataFormatString** (applied with String.Format to the value), **ConvertEmptyStringToNull** (used when saving a blank entry), and **ApplyFormatInEditMode**. See "DisplayFormatAttribute: Rules to convert from value to a string".

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

DisplayColumnAttribute – Specifies which column of the table to display (in filters or in foreign key links). By default, Dynamic Data uses the first column of type string that it finds. It also lets you change the default sorting, to specify a different column and sort order. See "DisplayColumnAttribute: Selecting a column name for foreign keys".

CacheListAttribute – Used on Enumerated and ForeignKey columns to retain their lists in the Application Cache instead of recalling their elements from the database with each page request. See "CacheListAttribute: Caching Lists for Enumerated and ForeignKeys".

ScaffoldColumnAttribute – Determines if a column is added by the automatic scaffolding feature to the user interface. By default, all columns supplied by the database are scaffolded. You use this to remove selected columns. All custom properties are omitted unless you use this to include them. See "ScaffoldColumnAttribute: Determining if Scaffolding adds a Column".

ScaffoldTableAttribute – Lets you omit a table when using the automatic scaffolding feature to build the user interface. By default, all tables supplied by the database are scaffolded. See "ScaffoldTableAttribute: Determining if a table appears in the UI".

SortableColumnAttribute – Determines if the column is sortable or not. Its used by the DynamicColumnTitle control to determine the kind of user interface to setup in a column title. See "SortableColumnAttribute: Determining sorting rules".

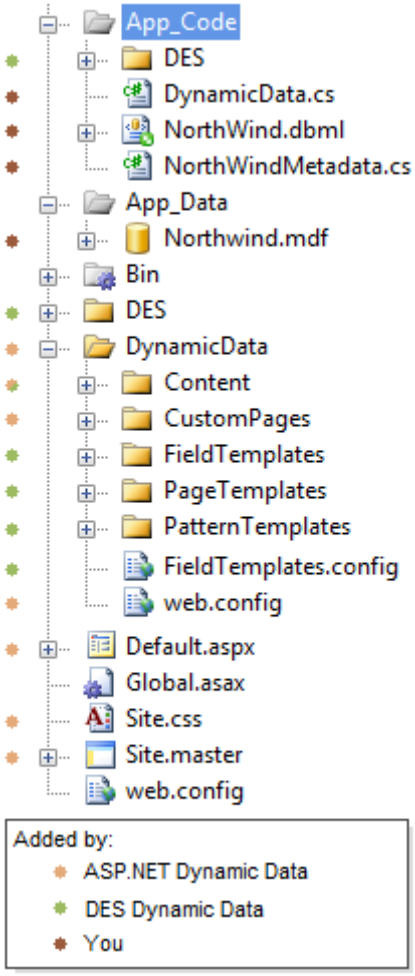
CategoryAttribute – Assists in selecting data fields by a group. It is used with the NamedStyles feature to change the style sheet based on fields of specific categories and by the Automatic Scaffolding to build lists of fields based on a group. See "CategoryAttribute: ".

TableRestrictionAttribute – Associates security restrictions with a table based on user roles. See "TableRestrictionAttribute: Deny access to tables by Role".

ColumnRestrictionAttribute – Associates security restrictions with a table based on user roles. See "ColumnRestrictionAttribute: Deny access to columns by Role".

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

Files and Folders



DynamicData folder – Installed with ASP.NET Dynamic Data to provide its default Page Templates and Field Templates. It usually resides in the web application root folder. Its subfolders are detailed below. DES's implementation of Dynamic Data adds the **FieldTemplates.config** file

DynamicData\FieldTemplates – The Field Templates. DES will replace the original Field Template files with its own. *DES's installation will backup the originals in a subfolder called "Original Files"*.

DynamicData\PageTemplates – The Page Templates. DES will replace the original Page Template files with its own. *DES's installation will backup the originals in a subfolder called "Original Files"*.

DynamicData\CustomPages – For more Page Templates that are specific to a table.

DynamicData\PatternTemplates – The PatternTemplates. DES ships with templates that mimic the GridView and DetailsView interfaces.

DynamicData\Content – Support files for images and several user controls that are used by the Page Templates. Most are supplied by ASP.NET Dynamic Data. DES's Page Templates use them without modification. DES Dynamic Data adds its own files to this folder.

DynamicData\FieldTemplates\Ajax Control Toolkit – Field Templates and other Usercontrols that utilize controls of the Ajax Control Toolkit. This is optionally installed.

DynamicData\FieldTemplates\RadControls– The Field Templates for Telerik RadControls for ASP.NET AJAX that work with DES. (Telerik also has its own Field Templates. Those should not be used.) This is optionally installed.

Web application root folder – ASP.NET Dynamic Data installs several files that create an application that uses its scaffolding feature. DES does not modify these. In addition, the `Application_Start()` method of **Global.asax** is used. So you must have a **Global.asax** file in your web application.

App_Code folder – Used with Web Site projects (where you do not have a Visual Studio project file) to hold source code files that must be compiled into your application. ASP.NET Dynamic Data needs several source code files: those created by LINQ to SQL or ADO.NET Entity Framework; the classes that correspond to the EntityTable classes where you define attributes; and optionally code called from `Application_Start()` to initialize Dynamic Data.

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

How DES enhances ASP.NET Dynamic Data

Peter's Data Entry Suite enhances the ASP.NET Dynamic Data toolset extensively to assist all aspects of its use.

Click on any of these topics to jump to them:

- Summary of major features
- Controls
- Field Templates
- PatternTemplates
- PageTemplates
- Attributes

Summary of major features

DES Dynamic Data enhances almost every aspect of ASP.NET Dynamic Data. It does not introduce any improved Data Source objects, but everything else has been considered for enhancement.

Business rules

The Attribute classes that you assign to the EntityTable Metadata classes have many improvements. You can also programmatically define business rule values at runtime that are specific to the situation.

- Validation rules – DES Dynamic Data uses DES's Validation Framework which is a much more powerful toolset than the native validator web controls. There are many more validator rules available and as a result, the ValidationAttributes have been overhauled and expanded. DES Dynamic Data supports validators that compare two columns, such as a start and end date.
- Scaffolding rules – The "automatic scaffolding" engine has been replaced to handle more cases and allow programmatic definition of the list of fields. The ScaffoldColumnAttribute handles additional cases.
- Datatypes – DES Dynamic Data recognizes many new data types and knows how to automatically use them to setup validating text for the data type.
- Security rules – Associate restrictions based on user roles to tables and columns. The user interface changes as a restricted feature is requested, whether to hide the feature or provide an alternative. See "Setup Security by User Roles".
- Character set rules – Use the CharacterSetAttribute to assign characterset limits to text type fields. When used, Field Templates can setup a FilteredTextBox and associated CharacterValidator.
- Enumerated type rules – Use the EnumeratedAttribute and EnumeratedStringAttribute to define a specific list of values allowed in an integer and string type field. With these, your user interface can offer a dropdownlist that maps text shown to the user to the internal string or integer stored in your tables.
- Sorting rules – Use the SortableColumnAttribute to identify if a column can be sorted or not. When sortable, column titles will automatically offer a sorting interface.
- Column categories – Use the CategoryAttribute on columns to assign one or more category names with the column. The categories can be used by automatic scaffolding to build the appropriate list of columns. They also can be used with the NamedStyles feature of PatternTemplates to change the appearance of the data, such as giving a bigger font to the category "Title".

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

Field Templates

All Field Templates have been overhauled, and there are many new ones.

- Validation now uses DES's validation framework.
- You never add actual validator controls to the Field Template. Instead, you add a single control, ColumnValidatorManager, to your Field Template and it uses the business rules to create the validators needed. This guarantees that your Field Templates always match the business rules.
- DES's controls from the Peter's Textboxes and Peter's Date and Time are used in many Field Templates, such as Integer_Edit.ascx uses IntegerTextBox and Date_Edit.ascx uses DateTextBox.
- Very little code is needed in each Field Template because of a rich set of classes to create your Field Templates. For example, Date_Edit.ascx inherits from the PeterBlum.DES.DynamicData.DateEditFTUC class.
- Customization of Field Templates can often be done without editing the actual Field Template. With the Customizer control on your form, you can define property values that will be applied automatically to the controls within the Field Template. For example, set the DateFormat to "Abbreviated" in the Customizer and Date_Edit.ascx will use that format.
- Field Templates have been created for many datatypes that aren't associated with DES's own textboxes, such as support for images, URLs, and files.

Webform Development

ASP.NET Dynamic Data primarily uses the Page Template to format a page. It offers automatic scaffolding to define the table's columns within a Page Template. It becomes more complex to avoid Page Templates or automatic scaffolding. The moment you need to customize the appearance of your list or single-record view, you must abandon scaffolding and start on a fresh form.

With DES Dynamic Data…

- Page Templates have been greatly simplified by removing most of the code. It now resides in an enhanced DynamicDataManager control, which knows how to create all of the event handlers for each type of databound control.
- Use PatternTemplates to describe how to output your data. They allow you to fully control all HTML generated while automating the setup of the DataBound control. You only need to specify a list of data fields to output. For example, the **DetailsView.ascx** and **GridView.ascx** PatternTemplate files mimic the appearance of the DetailsView and GridView controls respectively.
- Use the DynamicListView control instead of the ListView and GridView controls. It effectively merges the simplicity of setup offered by the GridView with the flexibility of the ListView by using very powerful PatternTemplates that know every part of a list-style user interface. Similarly, use the DynamicFormView control instead of the FormView and DetailsView controls.
- Use the DynamicButtons control to create the buttons shown in your DataBound control. It is very smart and flexible, showing buttons that apply and helping them communicate their "commands" to the DataBound control.
- Use the DynamicControlPanel control to provide paging, sorting, and other commands that manipulate the data shown in the DataBound control
- Use the DynamicColumnTitle control in column headings to offer both the correctly labeled title and sorting capabilities.

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

Controls

DES introduces new versions of DynamicControl, DynamicField, DynamicDataManager, and DynamicValidator. It also introduces new controls.

- **DynamicDataManager** – DES subclasses the `System.Web.DynamicData.DynamicDataManager` control to introduce functionality that simplifies to code within a Page Template or your own pages. Generally you will be using DES's `PeterBlum.DES.DynamicData.DynamicDataManager` control instead of the native control. See "DynamicDataManager Control and Adapters".
- **DynamicValidator** – You will not be using the native DynamicValidator. DES provides its own that conforms to the DES validation framework. In addition, its Field Templates and Page Templates create it for you automatically. You will rarely need to declare it. See "DynamicValidator Control".
- **DynamicField** – DES subclasses the DynamicField, which is used on GridView and DetailsView controls to display the Field Template for a specified column. Its enhancements include mode-specific UIHint and properties to assign to the Field Template. See "DES.DynamicField: Using Dynamic Data in GridView and DetailsView".
- **Customizer** – Provides page-level settings to the Field Templates and Page Templates, so you don't have to create new templates (or worse, edit existing ones that were correctly setup for other pages). The Customizer is where you define default error messages for validators, show currency or percent symbols, right align numeric textboxes, and much more. See "Customizer Control".
- **ColumnValidatorManager** – Creates all of the validators demanded by the ValidationAttributes and DataTypeAttributes. Add one to each Field Template that supports editing. See "ColumnValidatorManager Control".
- **DynamicListView** – A subclass of ListView that uses the PatternTemplate feature to mimic the GridView. While the GridView is an easy control to setup, it has a fixed HTML structure which imposes limitations in your user interface. Typically when the GridView's limitations effect your work, you switch to the ListView and have to reconstruct all of the HTML and controls that were built into the GridView. That's a lot of work!

  DynamicListView basically sets up the ListView's ItemTemplate and other templates, offloading the HTML formatting to the "GridView" PatternTemplate. You will modify the PatternTemplate or specify another one to adjust your user interface, all while still using the simplicity of scaffolding or a list of column names.

  The result is that DynamicListView is as easy to setup as GridView, and far more flexible. See "The DynamicListView Control".
- **DynamicFormView** – A subclass of FormView that uses the PatternTemplate feature to mimic the DetailsView. While the DetailsView is an easy control to setup, it has a fixed HTML structure which imposes limitations in your user interface. Typically when the DetailsView's limitations effect your work, you switch to the FormView and have to reconstruct all of the HTML and controls that were built into the DetailsView. That's a lot of work!

  DynamicFormView basically sets up the FormView's ItemTemplate and other templates, offloading the HTML formatting to the "DetailsView" PatternTemplate. You will modify the PatternTemplate or specify another one to adjust your user interface, all while still using the simplicity of scaffolding or a list of column names.

  The result is that DynamicFormView is as easy to setup as DetailsView, and far more flexible. See "The DynamicFormView Control".
- **DynamicPattern** – When using PatternTemplates, use the DynamicPattern control within the ListView and FormView control to load a PatternTemplate and specify its data field names and other configuration properties. (DynamicListView and DynamicFormView internally use it to load their PatternTemplates. GridView and DetailsView support its use within their TemplateField element.) See "The DynamicPattern Control".
- **DynamicPartsPatterns** – Used within a PatternTemplate file to describe all of the parts of a list-style and single record style control's user interface. "Parts" contain HTML and web controls. Those that generate the user interface for data fields contain DynamicControls that get their **DataField** property assigned from a DynamicPattern, DynamicListView, or DynamicFormView control. There are Parts for headers, footers, buttons, column titles and much more. Each Part can offer a list of alternatives for their output, called "NamedParts". That way you can use either the standard or Named Part to allow your DynamicPattern, DynamicListView, and DynamicFormView controls to build the right interface for the situation. See "The DynamicPartsPatterns Control".

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

- **DynamicControl** – Specifies a data field name that will load a Field Template in its location. It specify the exact Field Template desired or locate one by rules on the EntityTable MetaData class.

  This is the central control to Dynamic Data. It's used anywhere the data field will be shown on the page, although you may not need to add it because it's built into other controls and features: DynamicField, PatternTemplates, and automatic scaffolding.

  DES has subclassed the ASP.NET DynamicControl version to add a few capabilities. Either can be used within DES Dynamic Data. See "DES.DynamicControl: The heart of Dynamic Data".

- **DynamicLabel** – A companion control to DynamicControl. It specifies a data field name and converts it's name into a Label control. It is a subclass of the Label control, so it supports the **AssociatedControlID** property which lets the Label control use a <label for> tag to connect it to another HTML element. In this case, the associated control should be a DynamicControl.

  You will often use it within PatternTemplates as they will automatically assign the data field name to both a DynamicControl and associated DynamicLabel. See "The DynamicLabel Control".

- **DynamicButtons** control – Similar to the CommandField object used by GridView and DetailsView, this control generates the correct buttons based on the data entry mode. For ReadOnly, the buttons can be: New, Edit, Delete, Details, and Select. For Edit and Insert, the buttons can be Save and Cancel. These buttons direct the DataBound control to take a specific action. See "The DynamicButtons Control".

- **DynamicColumnTitle** – Used to form column titles on a list of records, using the ListView or DynamicListView. Column titles may initially appear to be a label and therefore should use the DynamicLabel. However, if your list supports sorting on a column title, this control knows how to handle that user interface, adding click to sort capability. It also knows when not to support sorting, based on the SortableColumnAttribute defined in the EntityTable Metadata classes. See "The DynamicColumnTitle Control".

- **DynamicTagStyles** control – Used within PatternTemplates to create HTML tags that have styles and other attributes from the Named Styles supplied by the DynamicPattern, DynamicListView, or DynamicFormView controls. See "The DynamicStylesTag Control".

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

Field Templates

- Replacements for most of the existing Field Templates, to support the DES Validation Framework. They are in the **[web app]/DynamicData/FieldTemplates** folder.

  Note: DES Dynamic Data's Field Templates have a different architecture from the originals, and use DES's validators. If you have customized Field Templates, you will modify those from DES. The original Field Templates are backed up for your reference in the [web app]/DynamicData/FieldTemplates/Original Files folder.

- Field Templates for each of its datatype oriented textboxes: Integer, Decimal, Currency, Percent, Date, Time, Duration, MonthYear, and Anniversary. There are specialized Field Templates too: Calendar_Edit.ascx uses the DES Calendar control for Dates; Measurement for integers and decimals that reflect a measurement; and for Telerik RadControls users, templates built upon many of their data entry controls. See "Field Templates in the DES/FieldTemplates folder".
- Feature rich base classes for Field Templates where you only need to supply the actual web controls. All of the logic for data binding is handled. So is selecting the right validators and applying settings from the Customizer control. This greatly simplifies creating new Field Templates and eliminates bugs. See "Creating and modifying Field Templates".
- You will not need to add any Validators to your Field Templates. Instead, you add the ColumnValidatorManager Control which uses the ValidationAttributes and other information to create your validators.
- Uses a new FieldTemplateFactory class (DESFieldTemplateFactory) to lookup Field Templates according to rules you define in the **[web app]/DynamicData/FieldTemplates.config** file. It provides aliases to map the requested template name to a new name and the ability to look for Field Templates in several folders. See "FieldTemplateFactory: Selecting the correct Field Template file".
- Runtime customization of Field Templates. Use the ICustomizeFieldTemplate interface to change the appearance of controls without editing the Field Template files or make modifications that vary based on the current data. See "Implementing the ICustomizeFieldTemplate Interface".
- Extended UIHintAttribute and DynamicField control can set the properties on Field Templates and their child controls based on the current mode (readonly, edit, or insert). See "Setting Properties on the Field Templates".
- Elimination of code behind files. While code behind has a purpose, when you need to duplicate web forms, they can cause problems, especially with regard to namespaces. Many times you end up with two web forms that have the same class and namespace, causing compilation errors. Since DES moves most of the code into its base classes, the remaining code on a Field Template is so small, that the benefits of single file outweigh the benefits of code behind.

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

PatternTemplates

PatternTemplates are an important tool when creating a Dynamic Data user interface. They are special UserControls that defines a pattern of controls and HTML, including DynamicControls (DynamicData's control that converts a column's name into a Field Template). Web page development frequently uses patterns, such as the HTML for a table row with two columns, where the first column contains a field name and the other is the field's data.

Patterns can be very complex. A GridView control is really a pattern, with a table tag enclosing numerous table rows of data. DES includes PatternTemplates that mimic the GridView and DetailsView. PatternTemplates have a significant advantage over these controls: you have full control over the HTML they generate. As a result, you will likely use the DynamicListView and DynamicFormView controls instead of GridView and DetailsView, because of this flexibility. DynamicListView and DynamicformView are introduced in DES Dynamic Data and are built around PatternTemplates.

> *Note: An upcoming release of ASP.NET Dynamic Data introduces "Entity Templates". That is a very simplistic implementation of PatternTemplates. While you can use Entity Templates within DES Dynamic Data, you will find PatternTemplates a better toolset.*

PatternTemplate files are stored in **[web app]/DynamicData/PatternTemplates**.

For examples using DynamicListView and DynamicFormView::

[C#]

[DES Product Folder]/DynamicData/CS/Samples/Using PatternTemplates

[VB]

[DES Product Folder]/DynamicData/VB/Samples/Using PatternTemplates

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

PageTemplates

These files are in the **[web app]/DynamicData/PageTemplates** folder.

- GridView and DetailsView have a fixed HTML output, limiting your creativity. DES Dynamic Data has replaced them with DynamicListView and DynamicFormView to take advantage of PatternTemplates, allowing complete control over the HTML output while still supporting scaffolding. If you prefer the GridView and DetailsView Page Templates, they are still available within the Page Template files. Follow directions in those files for switching to the GridView and DetailsView Page Templates. See "Pattern Templates: Reusable formatting for DynamicControls".
- Uses DataBound Control Adapter classes to simplify and enhance them. These classes are specific to the DataBound controls (GridView, FormView, etc), knowing exactly how to set them up. As you edit Page Templates, you do not have to worry about bugs created by incorrectly setup event handlers or code lost during editing because that code is in a separate compiled class. See "DynamicDataManager Control and Adapters".

  Note: DES Dynamic Data's Page Templates have a different architecture from the originals. If you have customized Page Templates, you will modify those from DES. The original Page Templates are backed up for your reference in the [web app]/DynamicData/PageTemplates/Original Files and [web app]/DynamicData/CustomPages/Original Files folder.

- Applies DES's own DynamicDataManager and ValidationSummary.
- You have much more control over automatic scaffolding (where Dynamic Data determines a list of columns for the default Page Templates). You can augment or even replace the list of columns, create the field objects and even assign property values for the Field Templates they retrieve. See "Customizing Automatic Scaffolding".
- Use the ICustomizeDataBoundControl interface from your Master Page to modify events and properties on the DataBound controls at runtime. Avoids modifying the actual Page Templates. See "Implementing the ICustomizeDataBoundControl Interface".
- Elimination of code behind files. While code behind has a purpose, when you need to duplicate web forms, they can cause problems, especially with regard to namespaces. Many times you end up with two web forms that have the same class and namespace, causing compilation errors.

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

Attributes

- There are ValidationAttributes for many validators within the DES Validation framework. See “ValidationAttributes: Applying Constraints on Columns”.
- The native ASP.NET Validation framework cannot apply its predefined validators to columns in different naming containers. Field Templates are separate naming containers. So the CompareValidator cannot be setup to compare two columns. The DES implementation fully supports multiple column validation through its new ValidationAttributes for CompareTwoFields, Difference and more.
- DES introduces the “DependencyAttribute”. The Database Developer applies it to columns where another column determines if this column is active or not. For example, a date column may not be available until a boolean column (represented by a checkbox) is true (checked). The dependency has two effects on the user interface. It disables (or hides) the data entry control and it disables its associated validators. See “DependencyAttributes: Relationships between columns”.
- DES introduces more datatypes. For example, the Measurement datatype is for any type of measurement, like gallons, inches, decibels, etc. The Database Developer can apply it to any integer or decimal column. They also define the labels for the type in singular, plural and abbreviated forms. The labels later appear in the Measurement Field Template automatically.
- DES introduces the TableRestrictionAttribute and ColumnRestrictionAttribute, which associate security restrictions by user role with tables and their columns. See “Setup Security by User Roles”.
- The Database Developer can identify a character set for a textual column by using the DES.CharacterSetAttribute. The Field Templates will adapt to that characterset, filtering out illegal characters as they are typed.
- When defining validators, sometimes they need to be created based on live data. DES lets your database developers create a ValidationAttribute based on that row by implementing the ICustomizeColumn interface in their EntityTable MetaData Class or IUICustomizeColumn interface on their web form.
- The MetaColumn class must never be edited on the **Field Template.Column** property. Yet it has some properties that you may want to edit while building the page: IsRequired, IsReadOnly, DefaultValue, MaxLength, ValidationAttributes, and DependencyAttributes. The **Field Template.ModifiableColumn** property lets you override the defaults for each of these properties by implementing the ICustomizeColumn or IUICustomizeColumn interfaces.
- Validating data types often requires more rules that just "test that it is a [name your type] value". For example, currencies may not allow negative numbers or a decimal field should never have more than 3 decimal places. DES's validators already know about these extra rules. The DES-supplied ValidationAttributes let the database developer dictate them.
- The DES.EnumeratedAttribute and DES.EnumeratedStringAttribute introduce support for enumerated types on integer and string columns.
- When the Field Template uses a ListBox or DropDownList to show items of an Enumerated type or ForeignKey, you can cache that list using the CacheListAttribute.
- There are enhanced versions of the DisplayNameAttribute, DescriptionAttribute, and UIHintAttribute.

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

FAQs

I don't recognize a lot of terms in this documentation, like EntityTable Metadata and DynamicControl. Where can I learn more about them?

See "The Elements of Dynamic Data".

Where can I find sample code files?

[DES Product Folder]\Dynamic Data*language*\Samples

Where do I edit the appearance of a data field?

Field Templates encapsulate the appearance of a data field for each specific data type. A Field Template is a UserControl file found in the **~/DynamicData/FieldTemplates** folder. The file name indicates the data type. Since the user interface changes depending on if the data entry mode is readonly, edit, or insert, there can be separate Field Template files for each mode. See "Field Templates: The user interface for columns".

I have added a ValidationAttribute to the EntityTable Metadata. What do I modify in Field Templates to support it?

You don't need to modify the Field Template so long as it has a ColumnValidatorManager control. That control's job is to convert your ValidationAttributes into Validator controls.

How do I override the default Field Template for my data fields?

There are several ways:

- On the EntityTable metadata, assign the UIHintAttribute and specify the new Field Template name (excluding the file extension and mode parts of the name.)
- On the DynamicControl, assign the **UIHint** property. Specify the new Field Template name (excluding the file extension and mode parts of the name.)
- On the DataFieldInPattern object, assign the **UIHint** property. Specify the new Field Template name (excluding the file extension and mode parts of the name.)
- If you want to globally change the Field Template used by a specific datatype and mode, edit the `<aliases>` node within **FieldTemplates.config** file.

Can I use other third party controls with DES Dynamic Data's Field Templates?

Yes. In fact, DES Dynamic Data comes with Field Templates for Telerik RadControls for ASP.NET AJAX and the AJAX Controls Toolkit. Third party data entry controls must be supported by the DES Validation Framework. See the **Using Third Party Controls.pdf** document.

There are many DataSource controls. Which should I use?

The web form requires a DataSource control. It handles the CRUD ("create, read, update, delete") actions. There are four DataSources that support Dynamic Data: LinqDataSource, EntityDataSource, DomainDataSource, and EntityDAODataSource. Your choice depends on what framework you choose to communicate with your data storage (like LINQ to SQL) and if you are enforcing a strong separation of concerns. For details, see "Choosing a Technology for your Business Logic".

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

There are numerous choices to setting up a DataBound Control that uses Dynamic Data. Which is best for me?

Let's start with what is the poorest choice: GridView and DetailsView controls. If you use them, you will be limited in how much you can customize its HTML output and setup style sheets. That leaves 3 other approaches: Page Templates; ListView and FormView controls; and DynamicListView and DynamicFormView controls.

- Page Templates define the user interface of an entire page, making the central feature of that page a list style or details style view of your data. By using ASP.NET Routing to configure the URLs, actions on those views – edit, insert, details, list – immediately load a different Page Template that is designed around that action. Finally, it involves all of the tables registered with your metadata. Page Templates make for quick setup but the application must take advantage of all of these.
- ListView and FormView controls allow complete control over the HTML output of your data. They do not come with any HTML output of there own, so you have to generate all aspects: headers, footers, how to make column titles sortable, buttons, etc. Fortunately, DES's tools can simplify the process because you can use PatternTemplates for individual data fields, the DynamicColumnTitles control, and DynamicButtons control which consolidate a number of elements into web controls. You can still use ASP.NET Routing and even automatic scaffolding if you use a PartsPatternTemplate.
- DynamicListView and DynamicFormView controls have all of the benefits of ListView and FormView controls (because they are direct subclasses) but eliminate most of the setup. That's because all of the HMTL formatting has already been specified in a PatternTemplate file. All you have to do is indicate which data fields to show. As a result, these controls are the recommended choice. (DES's Page Templates internally use them.)

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

Installing Dynamic Data Into Your Web Application

Note: As you work with DES Dynamic Data, you will quickly realize that you are working with two systems: the native ASP.NET Dynamic Data and the enhancements introduced in Peter's Data Entry Suite. This documentation will sometimes direct you to Microsoft's documentation where they cover the topic well.

ALERT: Dynamic Data requires a minimum of ASP.NET 3.5 SP 1 and Visual Studio 2008 SP 1.

There are several aspects to setting up your web application with Dynamic Data. They are covered in the next few sections.

- Getting the initial files in place, including ASP.NET's Dynamic Data files, DES's Dynamic Data Files, and preparing the `Application_Start()` method. This section covers these topics.
- Creating classes that represent your data. See "Creating EntityTable Classes".
- Applying Attributes to the classes that represent your data. See "Applying Attributes To Columns".
- Making the interface respect any security (login) restrictions. See "Setup Security by User Roles".

Click on any of these topics to jump to them:

- Adding ASP.NET Dynamic Data
- Adding DES Dynamic Data
  - Prerequisites for using DES Dynamic Data
- Initializing Dynamic Data at application startup
- Creating EntityTable Classes
- Setup Security by User Roles

How will you use Dynamic Data?

ASP.NET Dynamic Data can be used in several ways. You should review the options here, as the installation differs based on your approach.

- Dynamic Data Scaffolding Application – The Page Templates are used to build a database driving application that can explore and edit tables. This application also uses several support forms and files, including images, a master page and style sheets.
- Table Editor using a complete web form – Page Templates are complete web forms with their own master page that follow the tables retrieved by the DataSource. Unfortunately, they assume you will browse through multiple tables, which have been defined in your DatabaseContext class. For this approach, you will create a new web form, adding the necessary controls and setting up the DataSource to reflect the DataContext, TableName, and other query rules.
- Table Editor in an existing page – Often applications need the elements of Dynamic Data to be inserted into an existing page. For this approach, you will work within an existing form, adding the necessary controls and setting up the DataSource to reflect the DataContext, TableName, and other query rules. You may elect to use a UserControl to host the Dynamic Data elements of your form.

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

Prerequisites for using DES Dynamic Data

- ASP.NET 3.5 SP 1 and Visual Studio 2008 SP 1. It supports ASP.NET 4 and VS2010.

  Note: If using a prerelease of ASP.NET Dynamic Data, it must be in the SP 1 Beta or the May 23, 2008 release.

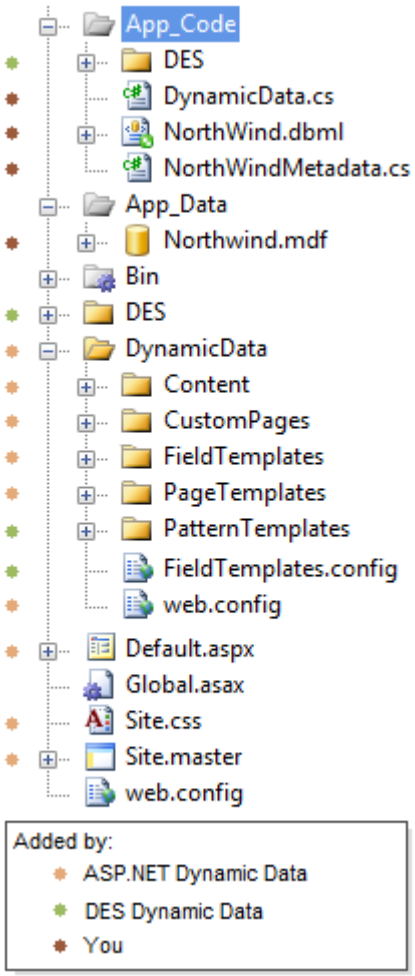
- If you do not have a license for the entire Suite of Peter's Data Entry Suite, you will still need several licenses:
  - Peter's Professional Validation – required. This implementation only uses the DES Validation Framework
  - Peter's More Validators – optional. Only if you use ValidationAttributes assocaited with validators in this module.
  - Peter's TextBoxes – required.
  - Peter's Date and Time – required if you use DateTime objects.
  - Peter's Interactive Pages – optional. Required when using DependencyAttributes.
  - Peter's Input Security – optional.

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

Adding ASP.NET Dynamic Data

ASP.NET Dynamic Data requires several files and folders in your web application. They are shown here:

File or Folder	Description
DynamicData	It usually resides in the web application root folder. Its subfolders, below, provide its content.
DynamicData\FieldTemplates	Contains the Field Templates native to ASP.NET Dynamic Data. *DES will replace these files.*
DynamicData\PageTemplates	Contains the Page Templates native to ASP.NET Dynamic Data. *DES will replace these files.*
DynamicData\CustomPages	For Page Templates that are specific to a table.
DynamicData\Content	Support files for images and several user controls that are used by the Page Templates. Supplied by ASP.NET Dynamic Data. DES's Page Templates use them without modification. DES adds its own style sheet files and images here.
Global.asax	This is a standard file for a web application. You will extend its `Application_Start()` method to initialize Dynamic Data.
DynamicData.cs or **DynamicData.vb**	A code file supplied with Dynamic Data that contains most of the code used by `Application_Start()`. You will modify this file to identify the DatabaseContext, FieldTemplateFactory, and Page Templates used by your application. It is usually located in the App_Code folder.
Site.css	The style sheet file used by the Page Templates and "Dynamic Data Scaffolding Application".
Site.master	The master page file referenced by the Page Templates.
Default.aspx	The startup page for the "Dynamic Data Scaffolding Application". It lists the tables defined by your DatabaseContext and redirects to Page Templates to view and edit the selected table.



All files in Dynamic Data can be edited to make them conform to your needs. Go ahead and change the style sheets, Field Template, Page Templates, etc. If you are not using the "Dynamic Data Scaffolding Application", remove the Default.aspx file.

See also http://msdn.microsoft.com/en-us/library/cc668159.aspx.

Click on any of these topics to jump to them:

- Using Visual Studio – For new applications only
- Using Visual Studio – For existing applications only
- Not Using Visual Studio

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

Using Visual Studio – For new applications only

1. Start Visual Studio.
2. In the **File** menu, click **New**, and then click **Web Site**.

   The **New Web Site** dialog box is displayed.
3. Under **Visual Studio installed templates**, select one of these:
   - **Dynamic Data Web Site** – When using LINQ to SQL.
   - **Dynamic Data Website Wizard** – Also when using LINQ to SQL. Uses a wizard to assist you in adding the Data Context (classes derived from your database).
   - **Dynamic Data Entities Web Site** – When using ADO.NET Entity Framework.
4. Set up the **Location** and **Language** fields.
5. Click **OK**.
6. If you have selected the Wizard, follow its prompts.

Visual Studio creates the folders and the files for a Dynamic Data Web site with the files listed above.

DES Dynamic Data has not yet been setup. Continue with "Adding DES Dynamic Data".

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

Using Visual Studio – For existing applications only

Visual Studio does not offer a way to add the Dynamic Data files, folders, and assemblies after you have an existing site, making this process somewhat complex.

1. Create a Starter Web App using the instructions below. Its contents will be used to setup your application.

   [C#]

   Unzip the file **[DES Product Folder]\DynamicData\CS\Starter Web App.cs.zip** into the same folder.

   [VB]

   Unzip the file **[DES Product Folder]\DynamicData\VB\Starter Web App.vb.zip** into the same folder.

2. Start Visual Studio.

3. In the **File** menu, open your existing web application where you want Dynamic Data files.

4. *ASP.NET 3.5 only.* Open your application's **web.config** file and add these elements, if not already present. You can copy them from the Starter Web App's **web.config** file, if you like. That file has comments prior to any lines that need to be copied.

   ASP.NET 3.5 only. In the `<system.web>` section:

```
    <compilation debug="true">
        <assemblies>
            <add assembly="System.Core, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=B77A5C561934E089"/>
            <add assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=31BF3856AD364E35"/>
            <add assembly="System.Web.Abstractions, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=31BF3856AD364E35"/>
            <add assembly="System.Web.Routing, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=31BF3856AD364E35"/>
            <add assembly="System.ComponentModel.DataAnnotations, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=31BF3856AD364E35"/>
            <add assembly="System.Web.DynamicData, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=31BF3856AD364E35"/>
            <add assembly="System.Data.DataSetExtensions, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=B77A5C561934E089"/>
            <add assembly="System.Xml.LINQ, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=B77A5C561934E089"/>
            <add assembly="System.Data.LINQ, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=B77A5C561934E089"/>
            <add assembly="System.Data.Entity, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=B77A5C561934E089"/>
        </assemblies>
        <buildProviders>
<!--- DYNAMIC DATA USES THIS WHEN USING THE ENTITY FRAMEWORK -->
            <add extension=".edmx"
                type="System.Data.Entity.Design.AspNet.EntityDesignerBuildProvider"/>
        </buildProviders>

    </compilation>

    <pages>
        <controls>
            <add tagPrefix="asp" namespace="System.Web.UI"
                assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
                    PublicKeyToken=31BF3856AD364E35"/>
            <add tagPrefix="asp" namespace="System.Web.DynamicData"
                assembly="System.Web.DynamicData, Version=3.5.0.0, Culture=neutral,
                    PublicKeyToken=31BF3856AD364E35"/>
            <add tagPrefix="asp" namespace="System.Web.UI.WebControls"
                assembly="System.Web.Entity, Version=3.5.0.0, Culture=neutral,
                    PublicKeyToken=b77a5c561934e089"/>
        </controls>
    </pages>

    <httpModules>
        <add name="UrlRoutingModule" type="System.Web.Routing.UrlRoutingModule,
                System.Web.Routing, Version=3.5.0.0, Culture=neutral,
```

```
            PublicKeyToken=31BF3856AD364E35"/>
</httpModules>
```

In the `<system.webServer>` section:

```
<modules runAllManagedModulesForAllRequests="true">
    <remove name="UrlRoutingModule"/>
    <add name="UrlRoutingModule" type="System.Web.Routing.UrlRoutingModule,
            System.Web.Routing, Version=3.5.0.0, Culture=neutral,
            PublicKeyToken=31BF3856AD364E35"/>
</modules>
<handlers>
    <add name="UrlRoutingHandler" preCondition="integratedMode" verb="*"
            path="UrlRouting.axd" type="System.Web.Routing.UrlRoutingHandler,
            System.Web.Routing, Version=3.5.0.0, Culture=neutral,
            PublicKeyToken=31BF3856AD364E35"/>
</handlers>
```

5. Copy the files and folders from the Starter Web App. The approach depends on whether you are using a Web Site Application or Web Project Application (which has a project file).

Web Site Application

a. Using the Windows Explorer, view the Starter Web App's root folder.

b. Copy these files and folders to your web application's root folder:

ALERT: Avoid overwriting any files in your web application that you wish to preserve. Instead, rename the DynamicData files and make the necessary corrections as stated here.

- **DynamicData** (folder) – DES Dynamic Data replaces the contents of Page Templates and Field Templates with its own files. (It copies the existing files to a subfolder called "Original files".
- **Site.css** – If one already exists in your real application, locate the `<link>` tag in Site.master and change it to the new name.
- **Site.master** and its **codebehind file** – If one already exists, update the `<%@ Page %>` tag of Page Templates to refer their **MasterPageFile** property to the new name.
- **Default.aspx** and its **codebehind file** – If one already exists, do not replace it unless you are not using it. Instead, create a differently named web form with the same HTML and code behind, but using a different class and namespace.

c. In **Solution Explorer**, right click on the Web Site node and choose **Refresh** to see these files and folders.

d. If its not already declared, add the **App_Code** folder by right clicking on the project node and selecting **Add ASP.NET Folder**; **App_Code**.

e. Add the **DynamicData.cs** or **.vb** file to your **App_Code** folder. Right click on the App_Code node in the Solution Explorer and choose **Add Existing Item**. Locate the file in the Starter Web App's **App_Code** folder.

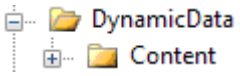
DES Dynamic Data has not yet been setup. Continue with "Adding DES Dynamic Data".

INSTRUCTIONS FOR WEB PROJECT APPS ARE ON THE NEXT PAGE

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

Web Project Application

a. *Create DynamicData and Content folders.* Using the Solution Explorer, create the following folders in the project, starting at the web application root node. Right click and use the **New Folder** command.



Note: DES Dynamic Data installation will add the Field Templates and Page Templates folders later.

b. *Populate the Content folders.* On the **Content** folder, right click and choose **Add Existing Item**. Locate the corresponding folder within the Starter Web App and add all of its files.

c. *Add the web.config file.* On the **DynamicData** folder, right click and choose **Add Existing Item**. Locate the corresponding folder within the Starter Web App and add the **web.config** file it contains.

d. On the web application's root node, choose **Add Existing Item**. Locate the Starter Web App's root folder and add the following files:

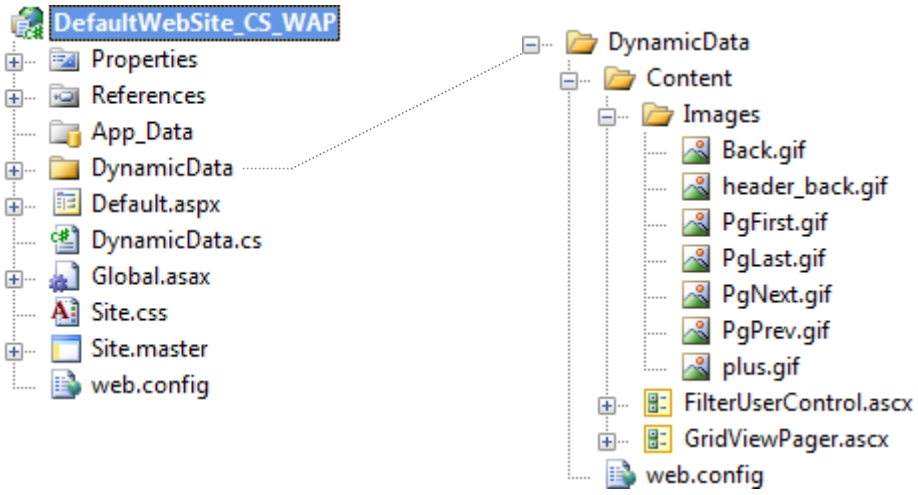
ALERT: Avoid overwriting any files in your web application that you wish to preserve. Instead, rename the DynamicData files and make the necessary corrections as stated here.

- **Site.css** – If one already exists in your real application, locate the `<link>` tag in **Site.master** and change it to the new name.
- **Site.master** and its **codebehind file** – If one already exists, see "Renaming the Master Page".
- **Default.aspx** and its **codebehind file** – If one already exists, do not replace it unless you are not using it. Instead, create a differently named web form with the same HTML and code behind, but using a different class and namespace. See "Renaming the Entry Point Page".

e. On the web application's root node, choose **Add Existing Item**. Locate the Starter Web App's **App_Code** folder and add the **DynamicData.cs** or **.vb** file.

f. Right click on the newly added **DynamicData.cs** or **.vb** file and choose **Properties**. Set **Build** to `Compile`.

DES Dynamic Data has not yet been setup. Continue with "Adding DES Dynamic Data".



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

Not Using Visual Studio

1. Create a Starter Web App using the instructions below. Its contents will be used to setup your application.

   [C#]

   Unzip the file **[DES Product Folder]\DynamicData\CS\Starter Web App.cs.zip** into the same folder.

   [VB]

   Unzip the file **[DES Product Folder]\DynamicData\VB\Starter Web App.vb.zip** into the same folder.

2. *ASP.NET 3.5 only.* Open your application's **web.config** file and add these elements, if not already present. You can copy them from the Starter Web App's **web.config** file, if you like. That file has comments prior to any lines that need to be copied.

   ASP.NET 3.5 only. In the `<system.web>` section:

```
    <compilation debug="true">
        <assemblies>
            <add assembly="System.Core, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=B77A5C561934E089"/>
            <add assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=31BF3856AD364E35"/>
            <add assembly="System.Web.Abstractions, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=31BF3856AD364E35"/>
            <add assembly="System.Web.Routing, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=31BF3856AD364E35"/>
            <add assembly="System.ComponentModel.DataAnnotations, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=31BF3856AD364E35"/>
            <add assembly="System.Web.DynamicData, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=31BF3856AD364E35"/>
            <add assembly="System.Data.DataSetExtensions, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=B77A5C561934E089"/>
            <add assembly="System.Xml.LINQ, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=B77A5C561934E089"/>
            <add assembly="System.Data.LINQ, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=B77A5C561934E089"/>
            <add assembly="System.Data.Entity, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=B77A5C561934E089"/>
        </assemblies>
        <buildProviders>
<!--- DYNAMIC DATA USES THIS WHEN USING THE ENTITY FRAMEWORK -->
            <add extension=".edmx"
            type="System.Data.Entity.Design.AspNet.EntityDesignerBuildProvider"/>
        </buildProviders>

    </compilation>

    <pages>
        <controls>
            <add tagPrefix="asp" namespace="System.Web.UI"
                assembly="System.Web.Extensions, Version=3.5.0.0, Culture=neutral,
                    PublicKeyToken=31BF3856AD364E35"/>
            <add tagPrefix="asp" namespace="System.Web.DynamicData"
                assembly="System.Web.DynamicData, Version=3.5.0.0, Culture=neutral,
                    PublicKeyToken=31BF3856AD364E35"/>
            <add tagPrefix="asp" namespace="System.Web.UI.WebControls"
                assembly="System.Web.Entity, Version=3.5.0.0, Culture=neutral,
                    PublicKeyToken=b77a5c561934e089"/>
        </controls>
    </pages>

    <httpModules>
        <add name="UrlRoutingModule" type="System.Web.Routing.UrlRoutingModule,
                System.Web.Routing, Version=3.5.0.0, Culture=neutral,
                PublicKeyToken=31BF3856AD364E35"/>
    </httpModules>
```

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

In the `<system.webServer>` section:

```
<modules runAllManagedModulesForAllRequests="true">
    <remove name="UrlRoutingModule"/>
    <add name="UrlRoutingModule" type="System.Web.Routing.UrlRoutingModule,
            System.Web.Routing, Version=3.5.0.0, Culture=neutral,
            PublicKeyToken=31BF3856AD364E35"/>
</modules>
<handlers>
    <add name="UrlRoutingHandler" preCondition="integratedMode" verb="*"
            path="UrlRouting.axd" type="System.Web.Routing.UrlRoutingHandler,
            System.Web.Routing, Version=3.5.0.0, Culture=neutral,
            PublicKeyToken=31BF3856AD364E35"/>
</handlers>
```

3. Copy these files and folders from the Starter Web App's root folder to your web application's root folder:

   ALERT: Avoid overwriting any files in your web application that you wish to preserve. Instead, rename the DynamicData files and make the necessary corrections as stated here.

   - **DynamicData** (folder) – DES Dynamic Data replaces the contents of Page Templates and Field Templates with its own files. (It copies the existing files to a subfolder called "Original files".
   - **Site.css** – If one already exists in your real application, locate the `<link>` tag in **Site.master** and change it to the new name.
   - **Site.master** and its **codebehind file** – If one already exists, see "Renaming the Master Page".
   - **Default.aspx** and its **codebehind file** – If one already exists, do not replace it unless you are not using it. Instead, create a differently named web form with the same HTML and code behind, but using a different class and namespace. See "Renaming the Entry Point Page".

4. If its not already defined, add the **App_Code** folder to your web application's root folder.
5. Copy the **DynamicData.cs** or **.vb** file from the Starter Web App's **App_Code** folder to your web application's **App_Code** folder.

DES Dynamic Data has not yet been setup. Continue with "Adding DES Dynamic Data".

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

Adding DES Dynamic Data

DES will add additional files and folders to your web application. While it does not overwrite any existing file, some files supplied by ASP.NET Dynamic Data are no longer used.

File or Folder	Description
DynamicData	A container for most Dynamic Data files and folders.
DynamicData\FieldTemplates	The Field Templates supplied by DES. (There are no code behind files.) *Installation will copy existing Field Templates into the DynamicData\FieldTemplates\Original Files folder.*
DynamicData\FieldTemplates \RadControls	Optional. The Field Templates for Telerik RadControls for ASP.NET AJAX that work with DES. (Telerik also has its own Field Templates. Those should not be used.)
DynamicData\FieldTemplates \Ajax Control Toolkit	Optional. Field Templates using some controls from the AJAX Control Toolkit.
DynamicData\PageTemplates	The Page Templates supplied by DES. (There are no code behind files.) *Installation will copy existingPage Templates into the DynamicData\PageTemplates\Original Files folder.*
DynamicData\CustomPages	For Page Templates that are specific to a table. Subfolders are the names of tables. You will copy the original Page Templates here and modify them. *Installation will copy existing Page Templates into the DynamicData\CustomPages\Original Files folder.*
DynamicData\PatternTemplates	PatternTemplate are user controls that are used by the DynamicPattern, DynamicListView, and DynamicFormView controls. PatternTemplates allow full customization of the HTML output and greatly reduce the setup work.
DynamicData\Contents	DES adds style sheet files to this folder: DESDynamicDataControls.css, PatternTemplateNamedStyles.css and FixSiteforDD1.css. These are automatically used when needed.
DynamicData\FieldTemplates.config	Guides DES's own FieldTemplateFactory to locate Field Templates. By default, it looks in the DynamicData\DES\FieldTemplates folder and if not found there, in the DynamicData\FieldTemplates folder. It can be used to map a specific datatype to a template, as if you had a UIHintAttribute for a datatype.
Bin\PeterBlum.DES.DynamicData.dll	The DES assembly specific to Dynamic Data. It contains several new controls that can be added to your toolbox.

Before you add DES support, be sure that ASP.NET Dynamic Data is already established in your web application. See "Adding ASP.NET Dynamic Data".

See also "Prerequisites for using DES Dynamic Data".

DES simplifies Page Templates and Field Templates. You may find these tools will avoid editing DES's files.

- The Customizer control sets many properties on the Field Templates. See "Customizer Control".
- The DES.UIHintAttribute, DES.DynamicControl, and DES.DynamicField can set properties on the Field Template object and on the controls it contains. See "Setting Properties on the Field Templates".
- The ICustomizeFieldTemplate interface exposes the Field Template object for you to modify programmatically from your Master Page. See "Implementing the ICustomizeFieldTemplate Interface".
- The IFieldGeneratorGuidance interface augments the scaffolding feature so you don't have to create custom Page Templates for each situation. See "Customizing Automatic Scaffolding".

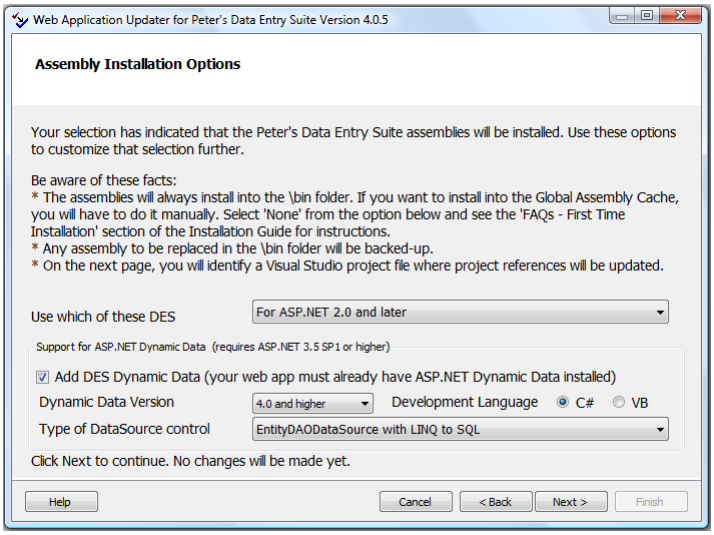
Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Steps to add DES Dynamic Data support

1. Run DES's **Web Application Updater** from in the **[DES Product Folder]**.
2. Use either the **First Time Installation** or **Update a Web Application** option, depending on if DES is already setup in your web application or not.
3. Proceed through the wizard. When you get to the **Assembly Installation Options** panel:
   - Mark **Add DES Dynamic Data**
   - Select which version of ASP.NET Dynamic Data you are using.
   - Select the language you use: **C#** or **VB**
   - Select the type of DataSource controls you will use.

     The EntityDAODataSource control is recommended. It is created by Peter Blum, is free, handles many technologies, and emphasizes separation of concerns between user interface and business logic.

     For details, see Versatile DataSources.



4. Continue through the wizard and let it install.
5. If you were using ASP.NET Dynamic Data prior to this installation, your existing Page Templates and Field Templates will be replaced. Backups will be made in the "Original Files" subfolder of each folder.
6. Modify your `Application_Start()` method to initialize Dynamic Data. This step includes creating a DESMetaModel object, calling `DESMetaModel.RegisterContext()`, and establishing Url Routing rules.

   See "Initializing Dynamic Data at application startup".
7. Add DES's PageManager control to **Site.master** and set its **AJAXFramework** property to "`MicrosoftAJAX`". This is required to let DES operate within the UpdatePanel declared on this file.

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

   Hint: The PageManager allows numerous ways to customize DES. In particular, the DefaultErrorFormatterSkinID can be used to change the ErrorFormatters used in validators contained through DES Dynamic Data. They normally use the Text ErrorFormatter which takes up a lot of space. Consider setting DefaultErrorFormatterSkinID to one of these: "image with alert box", "image with tooltip", "hyperlink with alert box", or "popupview".
8. Add the Customizer control. It is optional but very likely to be used. See "Customizer Control".

   ```
   <desDD:Customizer ID="Customizer1" runat="server" />
   ```

Note: DES Dynamic Data provides 3 additional style sheet files in the ~/DynamicData/Content/ folder. Like with all DES style sheet files, these will automatically be added to your page when needed.

SEE AN EXAMPLE OF THE MASTER PAGE ON THE NEXT PAGE

9. Consider modifying the DES Page Templates in the **DynamicData/PageTemplates** and **DynamicData/CustomPages** folder to match your earlier work. See "Customizing the standard PageTemplates" and "Creating Table-Specific PageTemplates".
10. Consider modifying DES Field Templates in the **DynamicData/FieldTemplates** to match your earlier work. See "Creating and modifying Field Templates".

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

Example: Minimal Master Page

Note: The tagprefix "desDD" maps DES Dynamic Data controls to the PeterBlum.DES.DynamicData assembly. Your web .config file has a line that establishes it in the <system.web><pages><controls> section.

```
<%@ Master Language="C#" CodeFile="Site.master.cs" Inherits="Site" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
    <link href="~/Site.css" rel="stylesheet" type="text/css" />
  </head>
<body class="template" >
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />

    <des:PageManager ID="DESPageManager" runat="server"
      AJAXFramework="MicrosoftAJAX" />
    <desDD:Customizer ID="Customizer1" runat="server" />

    <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">
    </asp:ContentPlaceHolder>
  </form>
</body>
</html>
```

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

Initializing Dynamic Data at application startup

The `Application_Start()` method is designed to initialize global values of your web application. It is declared in the **Global.asax** file. If you don't have a **Global.asax** file, see "When Missing the Global.asax File".

> While the `Application_Start()` method must execute your code, most web applications have been setup with the default code in a separate file, **App_Code/DynamicData.cs** or **.vb**. If so, your `Application_Start()` method should contain this one line:
>
> ```
> DynamicData.RegisterRoutes(RouteTable.Routes)
> ```

Dynamic Data uses `Application_Start()` to establish these values that operate your web application.

- Establish one or more DESMetaModel objects. Each DESMetaModel owns the remaining items. See "Create the DESMetaModel object".
- Call the MetaModel's `RegisterContext()` method to connect it to a data context. If you are using Versatile DataSources, see the section "Initializing Dynamic Data" in the **EntityDAODataSource.pdf** document.
- Establish the FieldTemplateFactory. DES uses a different FieldTemplateFactory than the one shipped with ASP.NET DynamicData to follow rules that you setup in the **DynamicData\FieldTemplates.config** file. It is established automatically when you use the DESMetaModel class.
- Establish how Page Templates are selected through "ASP.NET URL Routing". Choose from list and detail styles for the various modes: insert, edit, detail, and list. See "Set up ASP.NET URL Routing".
- When using Versatile DataSources with the Entity Framework, there is additional initialization. See "Using Versatile DataSources".

Click on any of these topics to jump to them:

- Create the DESMetaModel object
- Set up ASP.NET URL Routing
- Using Versatile DataSources

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

Create the DESMetaModel object

ASP.NET Dynamic Data uses the `System.Web.DynamicData.MetaModel` class to describe a database programmatically. This object is created in `Application_Start()` or `RegisterRoutes()`. You will switch to DES's subclass, `PeterBlum.DES.DynamicData.DESMetaModel`.

Note: The DESMetaModel class installs a more powerful FieldTemplateFactory class and introduces the RestrictionManager class, where role-based security is managed.

Make these edits

Note: The Web Application Updater attempts to do this for you automatically. It succeeds when it finds code that creates the MetaModel object either in Global.asax or DynamicData.cs or .vb.

The following shows how to update that code. If using the `RegisterRoutes()` method, the edits go there. Otherwise, they go directly in `Application_Start()`.

[C#]

```
using DES=PeterBlum.DES.DynamicData;
...
DES.DESMetaModel model = new DES.DESMetaModel();
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
...
Dim model As DES.DESMetaModel = New DES.DESMetaModel()
```

Warning: The model variable's class must be PeterBlum.DES.DynamicData.DESMetaModel, not System.Web.DynamicData in order to allow the RegisterContext method to fully run.

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

Set up ASP.NET URL Routing

Routes map URLs to specific web forms. They give Dynamic Data its ability to have URLs in this format:

http://domainname/table/record/action.aspx (for example **http://mydomain/products/30/edit.aspx**)

instead of using querystring parameters and declarating the actual form name.

Routes are `System.Web.DynamicData.DynamicDataRoute` objects attached to the MetaModel object. So they are setup in `Application_Start()` or `RegisterRoutes()`.

Note: Dynamic Data Futures (7/16/08 build) subclasses DynamicDataRoute with its PrettyDynamicDataRoute. This class is supported by DES Dynamic Data.

Please see the following Microsoft documentation for setting up Routes:

http://msdn.microsoft.com/en-us/library/cc488540.aspx#PageTemplates (Start at subtopic "Customizing the routes")

http://www.asp.net/learn/3.5-sp1/video-437.aspx (Video: How to enable Table Specific Routes)

Mapping PageTemplates to Actions

When using the Page Template system, the ASP.NET URL Routing features determine which page is selected. It knows a few things about the page requested:

- The Action – There are four actions:
  - Details - View the record in read-only mode
  - Edit – View the record and allow editing
  - Insert – Create a record
  - List – View a list of records
- The Table – The name of the table to be viewed. This is the original table name, not the one from an optional DisplayNameAttribute on the EntityTable Class.

The `System.Web.DynamicData.DynamicDataRoute` class converts the Action and Table into a URL. It also can reverse the process, using a properly formatted URL to get an Action and Table.

Your task is to create DynamicDataRoute objects in `Application_Start()` that define the URL pattern used for each Action. If you view the default code supplied with ASP.NET Dynamic Data, it demonstrates two configurations that use Page Templates. The configurations are:

- Use separate forms for all actions. That means use List.aspx, Details.aspx, Edit.aspx, and Insert.aspx.
- Use one form for all actions. That means use ListDetails.aspx.

Examples are shown in these locations:

[C#]

[DES Product Folder]/DynamicData/CS/Samples/App_Code/DynamicData.cs

[VB]

[DES Product Folder]/DynamicData/VB/Samples/App_Code/DynamicData.vb

Since both configurations are listed in the default code, you must uncomment out the DynamicDataRoutes you want to use and comment out the others.

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

Using Versatile DataSources

Versatile DataSources includes additional setup of `Application_Start()`.

When using ADO.NET Entity Framework

1. Entity Framework supports the many-to-many relationships. Confirm that your **/DynamicData/FieldTemplates** folder contains two additional Field Templates files, **ManyToMany.ascx** and **ManyToMany_Edit.ascx**.

   Also confirm that the **/App_Code/DES** folder contains either **Versatile DataSources Support.cs** or **.vb**.

   If they are not installed, you can copy them from:

   [DES product folder]/DynamicData/Other files for Dynamic Data Folder/CS or **VB**

2. Add the following lines to `Application_Start()` before the DESMetaModel is created:

   [C#]

```
YourObjectContextClass vObjectContext = new YourObjectContextClass();
PeterBlum.DataSources.Descriptors.EFDataContextDescriptors.RegisterEFDataContext(
  vObjectContext);
PeterBlum.DES.DynamicData.VDS.VDSManyToManyDataSourceManager.RegisterWithApp();

DES.DESMetaModel model = new DES.DESMetaModel();
```

   [VB]

```
Dim vObjectContext As YourObjectContextClass = New YourObjectContextClass()
PeterBlum.DataSources.Descriptors.EFDataContextDescriptors.RegisterEFDataContext(
  vObjectContext)
PeterBlum.DES.DynamicData.VDS.VDSManyToManyDataSourceManager.RegisterWithApp()

Dim model As DES.DESMetaModel = New DES.DESMetaModel()
```

   The *YourObjectContextClass* is the class generated by Entity Framework based on `System.Data.Objects.ObjectContext`. It hosts lists of your Entity classes. For example, the Northwind database may have been output to **NorthwindModel.Designer.cs** and has this declaration:

```
namespace NorthwindModel
{
   public partial class Entities : ObjectContext
   {
   }
}
```

   YourObjectContextClass is declared as `new NorthwindModel.Entities()` in the code above.

When using POCODataSource

You must create a PeterBlum.DataSources.DataContexts.POCODataContext class and register it with Dynamic Data in `Application_Start()`. Directions are found in the POCODataSource.pdf document in the section "Defining the POCODataSource class".

When registering it, the directions have you use a System.Web.DynamicData.MetaModel class. Instead use PeterBlum.DES.DynamicData.DESMetaModel. Here is a revised example:

```
using PeterBlum.DataSources.DataContext;
using PeterBlum.DataSources.DynamicData.Providers;
using DES=PeterBlum.DES.DynamicData;
…
DESMetaModel pocoTypesModel = new DESMetaModel();
POCODataContext dataContext = new MyPOCODataContext();
pocoTypesModel.RegisterContext(
 POCOModelProvider.CreateForMetaModel(dataContext));
```

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

When Missing the Global.asax File

DES provides a starter **Global.asax** file in its **[DES Product folder]\When Missing Global.asax**. Copy the file of the correct language to the root application of your web application.

When using a Web Application Project, right click on the root node of the project and select **Add New Item**. Then select Global Application.

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

Overview: Business logic on your tables

ASP.NET Dynamic Data separates the business logic from web forms. Business logic includes datatypes, validation, naming, and text formatting of any column in a table. When using ASP.NET Dynamic Data, the web form developer uses either a DynamicField or DynamicControl, assigning the column name to its **DataField** property. ASP.NET Dynamic Data does the rest, getting a Field Template to display and edit the column. It uses the business logic to select the Field Template and create the right validators.

This section shows the big picture on how to setup the business logic.

- Select a technology that represents your data as classes in your application. See "Choosing a Technology for your Business Logic".
- Create classes representing your tables. These classes are called EntityTables. See "Creating EntityTable Classes".
- Create classes corresponding to the EntityTables where you can add metadata (attributes on properties). These classes are called EntityTable Metadata Classes. See "Creating the EntityTable Metadata Classes".
- When using separation of concerns, you will also create classes for your queries. The EntityDAODataSource calls these EntityDAO classes. The DomainDataSource calls these the DomainDataContext. See the documentation for those controls for further information.
- Apply business logic to your EntityTable or their Metadata Classes. DES Dynamic Data has a rich toolset to cover most situations.
  - Apply attributes to columns to describe their business logic. See "Applying Attributes To Columns".
  - Apply attributes to the class declaration to cover table-level business logic. See "Applying Attributes To EntityTables".
  - Always invoke server side validation before the EntityTable class is saved. See "Invoking Validation prior to saving".
  - There are cases where you need to programmatically assign an Attribute based on the current situation. For example, add the RequiredAttribute (a ValidationAttribute) when the security role is for a data entry person but not for a manager. You will implement the `PeterBlum.DES.DynamicData.ICustomizeColumn` interface on your EntityTable. Its `CustomizeColumn()` method is passed a ModifiableColumn class, where you can modify many of the attributes. See "Implementing the ICustomizeColumn Interface".
  - There are cases where you want to create an actual Validator web control instead of letting DES Dynamic Data create it through a ValidationAttribute. You will implement the `PeterBlum.DES.DynamicData.ICreateValidatorWebControls` interface on your EntityTable. See "Defining Validators at Runtime: ICreateValidatorWebControls Interface".
  - You can add properties to your EntityTable that don't exist in the table. These become custom columns and are available to ASP.NET Dynamic Data. For example, your Customers table has separate columns for FirstName and LastName. Add a FullName property that returns FirstName + LastName. See "Creating Calculated and Custom Columns".
  - When you have security restrictions on tables or columns, setup the RestrictionManager and apply the TableRestrictionAttributes and ColumnRestrictionAttributes to define those restrictions. See "Setup Security by User Roles".

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

Choosing a Technology for your Business Logic

Select a technology that represents your data as classes in your application. There are two aspects to this.

- The system that communicates with the data storage. That includes ADO.NET, LINQ to SQL, ADO.NET Entity Framework, third party and your own custom solutions.
- Enforcing separation of concerns, where the business logic handles all data communications. The web form developer does not specify the logic of a query. In this case, the DataSource control plays an important role. Some DataSources handle separation of concerns. If you are interested in separation of concerns, use the DomainDataSource or EntityDAODataSource controls.

Here are the technologies and how the map to DataSource controls supported by Dynamic Data.

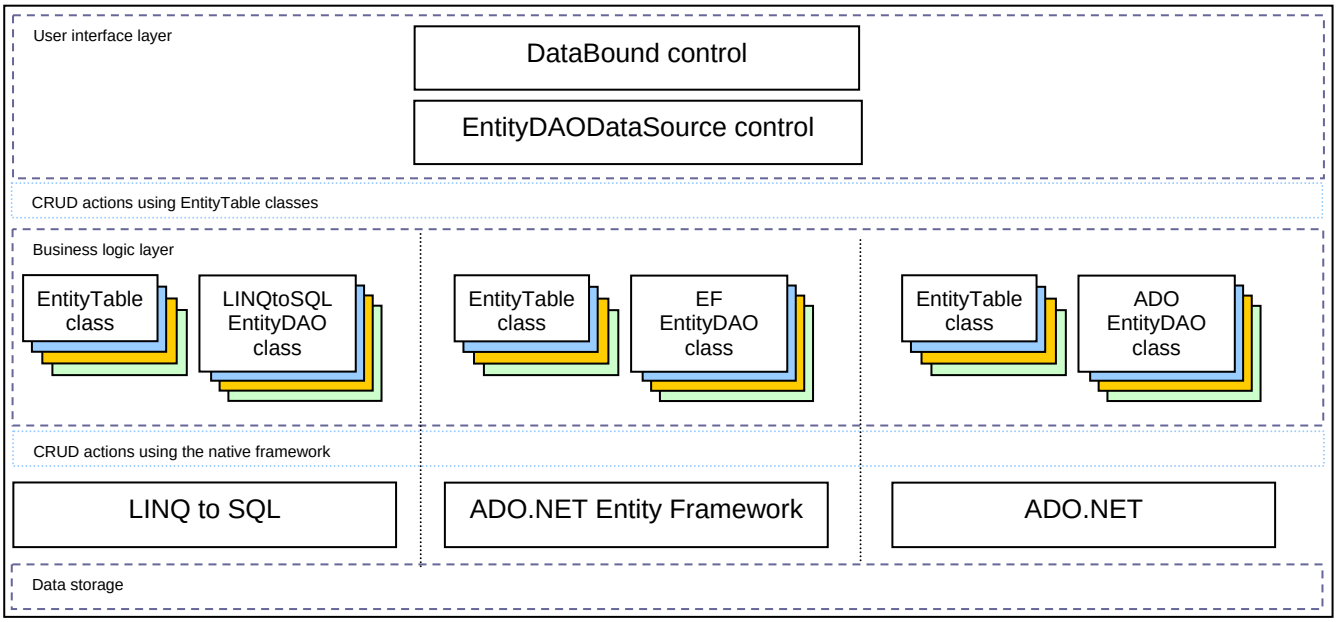
Technology	Separation of concerns	DataSource control
ADO.NET	No	None
ADO.NET	Yes	EntityDAODataSource
LINQ to SQL	No	LinqDataSource
LINQ to SQL	Yes	EntityDAODataSource and DomainDataSource
ADO.NET Entity Framework	No	EntityDataSource
ADO.NET Entity Framework	Yes	EntityDAODataSource and DomainDataSource
Custom Data Access Objects	Yes	EntityDAODataSource

Recommendation

Always use separation of concerns. It provides data access object classes that can be reused, such as for web services.

The best tool is Versatile DataSources, a free product from PeterBlum.com. It provides a methodology for developing Data Access Objects which host the CRUD ("create, read, update, and delete") methods. Its Data Access Objects support LINQ to SQL and ADO.NET Entity Framework EntityTable classes. It also works with ADO.NET and Data Access Objects that you may have already created.

Versatile DataSources provides the EntityDAODataSource control which strongly enforces separation of concerns. It also provides the POCODataSource which supports non-CRUD oriented activities.

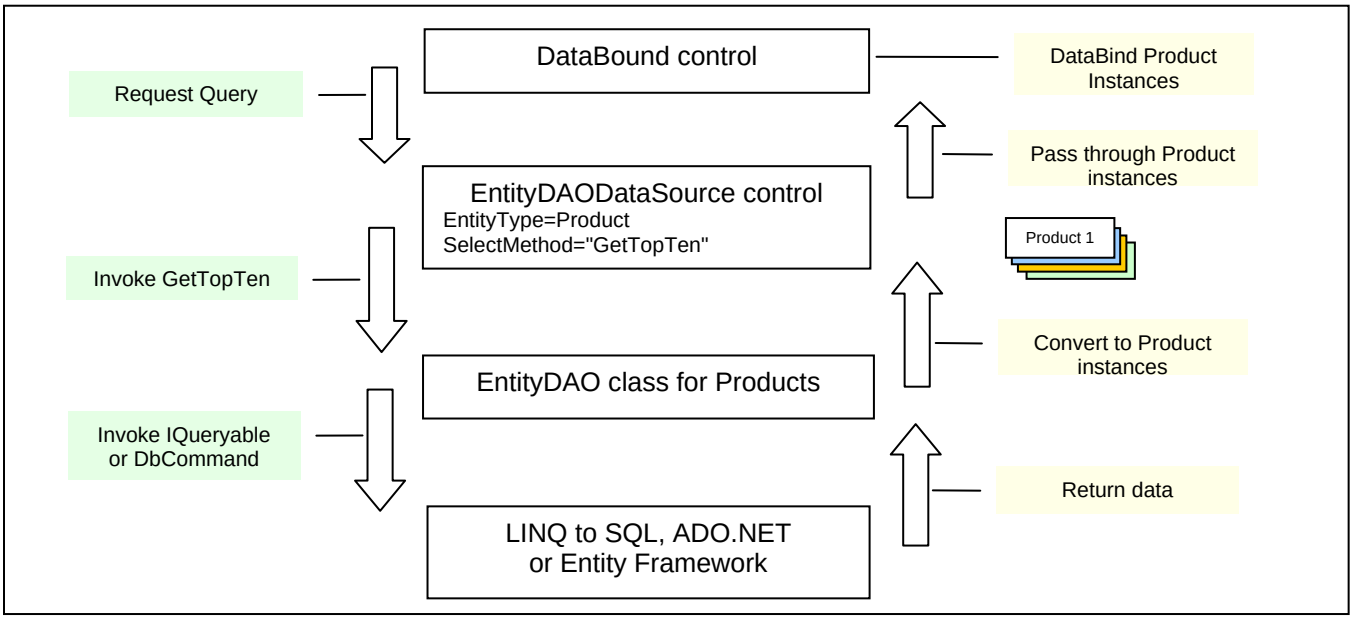


Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Query ("read") action on Products Entity class

The query action demonstrates how the separation of concerns is achieved when using Versatile DataSources.

The web form developer only provides two things to the EntityDAODataSource: The EntityTable class and the Select Method name. In this example, the Data Access Object for Products contains a method called GetTopTen.



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

Creating EntityTable Classes

The **Data Context** and **EntityTables** are classes that describe the structure of your data. Data Context identifies a list of related tables. EntityTables describe the columns of a single table. Dynamic Data is driven by these classes. When you create a MetaModel object, it is passed your Data Context class and from there Dynamic Data works out what tables, columns and associations (relationships) exist in your database.

When using LINQ to SQL or the ADO.NET Entity Framework, these classes are generated for you. (Instructions follow.) Note that these classes are not supposed to be edited by you. After all, any changes to your database may require regeneration of this code, effectively overwriting what was already there. So all edits – assigning attributes – are made to a separate set of classes that you create, called EntityTable Metadata classes. See "Defining business logic on each EntityTable".

When using ADO.NET or your custom Data Access Objects with Versatile DataSources, you may create these classes by hand. See the documentation of Versatile DataSources.

You can have multiple Data Contexts, each with a different list of EntityTables (or a different mixture from the same list). The Dynamic Data Scaffolding Application works with all tables from a single Data Context. Each Data Context must be assigned to its own MetaModel object in the `Application_Start()` method.

Click on any of these topics to jump to them:

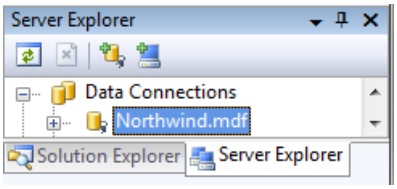
- Using LINQ to SQL
- Using ADO.NET Entity Framework
- Using Versatile DataSources
- Registering the DataContext with Dynamic Data

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

Using LINQ to SQL

These steps use Visual Studio 2008 or Visual Web Developer. They apply even if you are using Versatile DataSources with LINQ to SQL.

Credit: These steps are adapted from Microsoft's documentation, "Walkthrough: Creating a New Dynamic Data Web Site using Scaffolding".



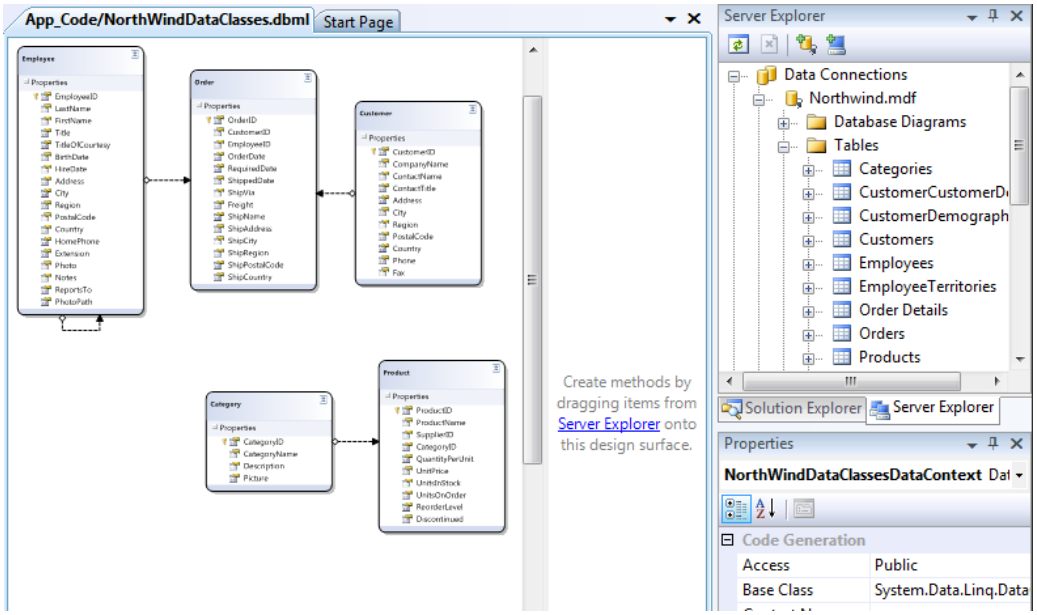
1. Your database should already have a Connection to your database defined. Visual Studio should list it in the Server Explorer view (Database Explorer in Visual Web Developer).
2. In Solution Explorer, right-click the App_Code folder and then click **Add New Item**.
3. Under Visual Studio installed templates, click **LINQ to SQL Classes**.
4. In the **Name** box, enter a name for the database model.

   Naming suggestion: *your database name*.dbml

   Your Data Context class will default to "*name*`DataContext`".
5. Click **Add**.

   The Object Relational Designer window is displayed.
6. In the Properties window, review the properties. If you prefer, set the **Context Namespace** property to a namespace for your Data Context class.
7. In the O/R Designer, click the **Server Explorer** link (**Database Explorer** in Visual Web Developer).
8. In Server Explorer (Database Explorer), under Data Connections, expand the database file node and then the Tables node.
9. Drag the desired tables into the Object Relational Designer window.

   Each table is represented as a class with the same name as the table itself. These classes are called EntityTables.



10. Close the O/R Designer window and let it save a ***name*.dbml** file. It will also create supporting files.

    You will actually use the one called ***name*.designer.cs** or **.vb** when working with Dynamic Data. View the designer.cs or .vb file to see the Data Context and EntityTable classes that were generated.

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

Your **web.config** file will now have an entry for your connection string in the `<connectionStrings>` section.

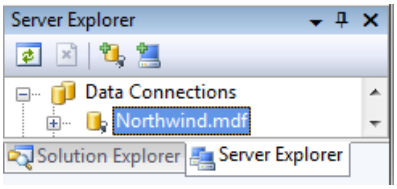
11. Open the file containing the application startup code, either in **Global.asax** or **App_Code\DynamicData.cs****.vb**.
12. If using Versatile DataSources, continue with "Using Versatile DataSources". Otherwise continue with "Registering the DataContext with Dynamic Data".

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

Using ADO.NET Entity Framework

These steps use Visual Studio 2008 or Visual Web Developer. They apply even if you are using Versatile DataSources with LINQ to SQL.

Credit: These steps are adapted from Microsoft's documentation, "Walkthrough: Creating a New Dynamic Data Web Site using Scaffolding".



1. Your database should already have a Connection to your database defined. Visual Studio should list it in the Server Explorer view (Database Explorer in Visual Web Developer).
2. In Solution Explorer, right-click the App_Code folder and then click **Add New Item**.
3. Under Visual Studio installed templates, click **ADO.NET Entity Data Model**.
4. In the **Name** box, enter a name for the database model.

   Naming suggestion: *your database name*.edmx

   Your Data Context class will default to "*name*`DataContext`".
5. Click **Add**.

   The Entity Data Model Wizard window is displayed.
6. Click **Generate from database**.

   This specifies that you want to generate the model from a database.
7. Under **Which data connection should your application use to connect to the database?**, select *your database name*.mdf from the list.
8. Make sure that the **Save entity connection settings in Web.config as** check box is marked and enter a name for the connection string name.

   Naming suggestion: *your database name*_DataContext
9. Click **Next**.

   The wizard displays a page where you can specify what database objects you want to include in your model.
10. Expand the node Tables and select the desired tables. Click **Finish**.

    The Entity Designer is displayed. You can close it.
11. If using Versatile DataSources, continue with "Using Versatile DataSources". Otherwise continue with "Registering the DataContext with Dynamic Data".

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

Using Versatile DataSources

This is a short overview of the techniques for establishing EntityType and DataContext classes when using Versatile DataSources. Versatile DataSources introduces a new class, the Entity Data Access Object ("EntityDAO") which holds the code all CRUD ("create, read, update, delete") actions. This allows strong separation of concerns where the web form developer never writes code that communicates with the database. Instead, they specify a method on the EntityDAO class and it handles the action.

1. Create the EntityTable classes.
   - When using LINQ to SQL, see "Using LINQ to SQL".
   - When using ADO.NET Entity Framework, see "Using ADO.NET Entity Framework".
   - For other sitations, see the section "Creating the Entity Classes" in the **EntityDAODataSources Users Guide**.
2. Create the DataContext classes.
   - When using LINQ to SQL or ADO.NET Entity Framework, you have already done this in step 1.
   - For other situations, see the section "Creating the DataContext Class" in the **EntityDAODataSources Users Guide**.
3. Create an EntityDAO class for each EntityTable class. See the sections "Creation the EntityDAO Classes" in the **EntityDAODataSources Users Guide**.
4. Continue with "Registering the DataContext with Dynamic Data".

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

Registering the DataContext with Dynamic Data

Once a Data Context class has been defined, it must be registered with a MetaModel object for Dynamic Data to use it.

The approach depends on if you are using Versatile DataSources or not. The MetaModel.RegisterContext() method takes different parameters in each case.

1. Open the file containing the application startup code, either in **Global.asax** or **App_Code\DynamicData.cs\.vb**.
2. Edit the line that calls RegisterContext()

Using Versatile DataSources

Change the first parameter to reflect this code, and specify the type of your Data Context:

[C#]

```
using PeterBlum.DataSources.DynamicData.Providers;
using DES=PeterBlum.DES.DynamicData;
…
DESMetaModel model = new DESMetaModel();
model.RegisterContext(
   EntityDAOModelProvider.CreateForMetaModel(
      typeof(YourDataContextType), true),
    new ContextConfiguration() { ScaffoldAllTables = false });
```

[VB]

```
Imports PeterBlum.DataSources.DynamicData.Providers
Imports DES=PeterBlum.DES.DynamicData
…
Dim model As DESMetaModel = New DESMetaModel()
model.RegisterContext( _
   EntityDAOModelProvider.CreateForMetaModel( _
      GetType(YourDataContextType), True), _
    New ContextConfiguration() With {.ScaffoldAllTables = False})
```

Not using Versatile DataSources

Change the first parameter to reflect the type of your Data Context:

[C#]

```
model.RegisterContext(typeof(yourDataContextType),
 new ContextConfiguration() { ScaffoldAllTables = false });
```

[VB]

```
model.RegisterContext(GetType(YourDataContextType),
 New ContextConfiguration() With {.ScaffoldAllTables = False})
```

3. If you are using the Dynamic Data Scaffolding Application or will have Page Templates drive your application, set **ScaffoldAllTables** to `true`.

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

4. If you need additional Data Contexts, repeat the steps, using different names for your dbml file and Data Context type. Then in step 2, create a new `PeterBlum.DES.DynamicData.DESMetaModel` object, cloning the existing code from `Application_Start()`. For example:

[C#]

```
using DES=PeterBlum.DES.DynamicData;
...
DES.DESMetaModel modelname = new DES.DESMetaModel();

modelname.RegisterContext(typeof(YourDataContextType),
   new ContextConfiguration() { ScaffoldAllTables = true });

RouteTable.Routes.Add(new DynamicDataRoute("{table}/{action}.aspx")
{
   Constraints = new RouteValueDictionary(
      new { action = "List|Details|Edit|Insert" }),
      Model = modelname
});
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
...
Dim modelname As DES.DESMetaModel = New DES.DESMetaModel()

modelname.RegisterContext(GetType(YourDataContextType),
   New ContextConfiguration() With {.ScaffoldAllTables = True})

RouteTable.Routes.Add(New DynamicDataRoute("{table}/{action}.aspx") With { _
   .Constraints = New RouteValueDictionary( _
   New With {.Action = "List|Details|Edit|Insert"}), _
    .Model = model})
```

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

Defining business logic on each EntityTable

This section describes how to apply your business logic to classes that represent your tables (the EntityTables).

Click on any of these topics to jump to them:

- Overview
- Steps for defining your business logic
- Creating the EntityTable Metadata Classes
- Applying Attributes To Columns
  - DataTypeAttributes: The DataTypes used by your Business logic
  - ValidationAttributes: Applying Constraints on Columns
- Applying Attributes To EntityTables
- Invoking Validation prior to saving
- Defining Validators at Runtime: ICreateValidatorWebControls Interface
- Customizing Metadata At Runtime
- Creating Calculated and Custom Columns
- Creating Instances of the EntityTable classes

C#

```
using DES=PeterBlum.DES.DynamicData;
...
[DES.DisplayName("Categories")]
public partial class Category
{
   [DES.Required()]
   [DES.DisplayName("Name")]
   public object CategoryName
      { get; set; }

   [DES.DataType(DataType.MultilineText)]
   [DES.RequiredDependency("CategoryName")]
   public object Description
      { get; set; }

   public void OnValidate(
      ChangeAction action)
   {
      if (CategoryNameExists())
         throw new ValidationException(
         "Duplicate category name.");
   }
}
```

VB

```
Imports DES=PeterBlum.DES.DynamicData;
...
<DES.DisplayName("Categories")> _
Public Partial Class Category

   <DES.Required()> _
   <DES.DisplayName("Name")> _
   Public Property CategoryName
      As Object
      Get and set code

   <DES.DataType(
      DataType.MultilineText)> _
   <DES.RequiredDependency(
      "CategoryName")>_
   Public Property Description
      As Object
      Get and set code

   Public Sub OnValidate(
      ByVal action As ChangeAction)
      If CategoryNameExists() Then
         Throw New ValidationException(
         "Duplicate category name.")
      End If
   End Sub
End Class
```

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

Overview

Your business logic belongs within the EntityTable classes. The user interface will request the "metadata" for EntityTable class from Dynamic Data and use it to construct an interface that matches your business logic.

Most of the business logic is defined using Attributes, objects that annotate the property or class definitions. For example:

C#

```
using DES=PeterBlum.DES.DynamicData;
...
 [DES.DisplayName("Categories")]
public class Category
{
   [DES.Required()]
   [DES.DisplayName("Name")]
   public object CategoryName
      { get; set; }

   [DES.DataType(DataType.MultilineText)]
   [DES.RequiredDependency("CategoryName")]
   public object Description
      { get; set; }
}
```

VB

```
Imports DES=PeterBlum.DES.DynamicData
...
<DES.DisplayName("Categories")> _
Public Class Category

   <DES.Required()> _
   <DES.DisplayName("Name")> _
   Public Property CategoryName
      As Object
      Get and set code

   <DES.DataType(
      DataType.MultilineText)> _
   <DES.RequiredDependency(
      "CategoryName")>_
   Public Property Description
      As Object
      Get and set code
End Class
```

ASP.NET Dynamic Data and DES Dynamic Data both offer you a library of Attributes. All are supported by DES Dynamic Data, although it is recommended to use DES's version where its available.

Here is an overview of the attributes you can assign to the property definitions:

- Refine the datatype with a **DataTypeAttribute** and its subclasses. The database stores data in very general types, like "Integer", "Text", and "Decimal". Your application uses them to store more specific types, like "Number of liters", "Phone number", and "Percent". This refinement helps select the right Field Template and determine some of the validation rules.
- Describe validation rules with subclasses of the **ValidationAttribute**. You can create ValidationAttributes on the fly, to supply rules that are based on the current data using the ICustomizeColumn and IUICustomizeColumn interfaces. You can even create actual DES Validator web controls, if needed, using the ICreateValidatorWebControls interface.
- Describe security restrictions on the column with the **ColumnRestrictionAttribute**.
- Describe a dependency relationship between columns using the **DependencyAttributes**. This is used when a column only needs data based on the state of another column. For example, "End Date" depends upon "Start Date" to be assigned. A DependencyAttribute will enable or disable the Required validator on the column and can visually disable or hide the data entry control.
- Override the default Field Template with the **UIHintAttribute**.
- Provide a friendly name for the column using the **DisplayNameAttribute**. Without it, Dynamic Data uses the name declared in your database.
- Provide a description to show in a tooltip or hint using the **DescriptionAttribute**.
- Provide an initial value when creating a new record with the **DefaultValueAttribute**.
- Provide string formatting rules to assist the Field Template as it converts a value to a string with the **DisplayFormatAttribute**.
- Determine if a column is included when scaffolding with the **ScaffoldColumnAttribute**.
- Select the name of the column in table that will be displayed as a foreign key with the **DisplayColumnAttribute**.
- Identify a column as sortable or not with the **SortableColumnAttribute**.

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

Here is an overview of the attributes you can assign to the property definitions:

- Describe security restrictions on the table with the **TableRestrictionAttribute**.
- Provide a friendly name for the table using the **DisplayNameAttribute**.
- Determine if a column is included when scaffolding with the **ScaffoldTableAttribute**.
- Specify which column to display in filters or in foreign key links with the **DisplayColumnAttribute**.

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

Steps for defining your business logic

At this point, you should have already taken these actions:

- Added ASP.NET Dynamic Data to your web application. See "Adding ASP.NET Dynamic Data".
- Adding DES Dynamic Data to your web application. See "Adding DES Dynamic Data".
- Setup Dynamic Data globals in `Application_Start()`. See "Initializing Dynamic Data at application startup".
- Created the Data Context and EntityTable classes. See "Creating EntityTable Classes".

1. When using LINQ to SQL or ADO.NET Entity Framework, create EntityTable Metadata classes. Your Attributes are assigned to these classes instead of the EntityTable classes themselves. (The EntityTable classes are generated by a code generator and can be overwritten.) See "Creating the EntityTable Metadata Classes".

   [C#]

   ```
   [MetadataType(typeof(CategoryMetadata))]
   public partial class Category
   {
   }

   public class CategoryMetadata
   {
   }
   ```

   [VB]

   ```
   <MetadataType(GetType(CategoryMetadata))> _
   Partial Public Class Category
   End Class

   Public Class CategoryMetadata
   End Class
   ```

2. Declare the namespace containing DES's attributes at the top of each source code file.

   [C#]

   ```
   using DES=PeterBlum.DES.DynamicData;
   ```

   [VB]

   ```
   Imports DES=PeterBlum.DES.DynamicData;
   ```

STEPS CONTINUE ON THE NEXT PAGE

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

3. Create properties in the EntityTable Metadata Class that have the same name (case sensitive) as the property in the EntityTable. These properties are actually placeholders. Their type and GET and SET code does not matter. So set property's type to object and setup the GET and SET elements as shown here:

   Note: You do not have to define every property. Those that will not need any attributes can be omitted.

   [C#]

```
public object propertyName    {get; set;}
```

   [VB]

```
Public Property propertyName As Object
   Get
      Return Nothing ' avoids a compiler warning when this block is empty
   End Get
   Set
   End Set
End Property
```

   [C#]

```
public class CategoryMetadata
{
   public object CategoryName { get; set; }
   public object Description { get; set; }
}
```

   [VB]

```
Public Class CategoryMetadata
   Public Property CategoryName As Object
      Get
         Return Nothing
      End Get
      Set
      End Set
   End Property
   Public Property Description As Object
      Get
         Return Nothing
      End Get
      Set
      End Set
   End Property
End Class
```

STEPS CONTINUE ON THE NEXT PAGE

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

4. Now you can add attributes to your properties. See ""Applying Attributes To Columns".

   DES provides equivalent attributes for most that come with ASP.NET Dynamic Data, as it attempts to enhance their features. To be sure that you are using DES's attribute, precede the attribute name with "DES.". For example:

   [C#]

```
[DES.DataType(DataType.Url)]
```

   [VB]

```
<DES.DataType(DataType.Url)>_
```

   The only attributes that do not need the "DES." because DES doesn't provide an equivalent are: DefaultValueAttribute ScaffoldTableAttribute, ScaffoldColumnAttribute, and DisplayFormatAttribute.

   Here are the attributes that you can assign to your Property placeholders:

- DataTypeAttributes: The DataTypes used by your Business logic
- ValidationAttributes: Applying Constraints on Columns
- DependencyAttributes: Relationships between columns
- UIHintAttribute: Customizing the Field Template used
- DisplayNameAttribute: Friendly column name
- DescriptionAttribute: A Hint for the user
- DefaultValueAttribute: Initial value when creating a new record
- DisplayFormatAttribute: Rules to convert from value to a string
- ScaffoldColumnAttribute: Determining if Scaffolding adds a Column
- SortableColumnAttribute: Determining sorting rules
- CategoryAttribute: Assigning Categories to Data Fields
- EditableAttribute: Prevent editing on certain columns
- ColumnRestrictionAttribute: Deny access to columns by Role
- CacheListAttribute: Caching Lists for Enumerated and ForeignKeys

   [C#]

```
using DES=PeterBlum.DES.DynamicData;
...
public class CategoryMetadata
{
   [DES.Required()]
   [DES.DisplayName("Name")]
   public object CategoryName { get; set; }

   [DES.DataType(DataType.MultilineText)]
   [DES.RequiredDependency("CategoryName")]
   public object Description { get; set; }
}
```

VB EXAMPLE IS ON THE NEXT PAGE

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

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
...
Public Class CategoryMetadata
   <DES.Required(), _
   DES.DisplayName("Name")> _
   Public Property CategoryName As Object
      Get
         Return Nothing
      End Get
      Set
      End Set
   End Property
   <DES.DataType(DataType.MultilineText), _
    DES.RequiredDependency("CategoryName")> _
   Public Property Description As Object
      Get
         Return Nothing
      End Get
      Set
      End Set
   End Property
End Class
```

5. If you could not find a ValidatorAttribute to meet your validation requirements, you have several choices:

   - Implement the ICreateValidatorWebControls interface to generate Validator web controls.
   - Create a new ValidationAttribute. See "How to: Customize Data Field Validation Using Custom Attributes", but subclass from PeterBlum.DES.ValidationAttribute instead of System.ComponentModel.DataAnnotations.ValidationAttribute.
   - Use the CustomValidationAttribute (requires ASP.NET 4). This attribute will only provide server-side validation. See "Writing server side only Validation Code within your EntityTable Classes".
   - If using ASP.NET 3.5, add code to individual properties and a more general method called during saving to perform that validation. Throw a ValidationException if an error is detected. See "Writing server side only Validation Code within your EntityTable Classes".

   See also, "How to: Customize Data Field Validation."

6. Apply Attributes to the class definitions on each EntityTable or EntityTable Metadata class. See "Applying Attributes To EntityTables".

   [C#]

```
[DES.DisplayName("Categories")]
public class Category
{

}
```

   [VB]

```
<DES.DisplayName("Categories")> _
Public Class Category

End Class
```

STEPS CONTINUE ON THE NEXT PAGE

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

7. Always add a validation method that evaluates all validation rules prior to saving. See “Invoking Validation prior to saving”.
8. Attributes contain static information. If you need any attribute’s value to be determined at runtime, use the ICustomizeColumn and IUICustomizeColumn interfaces. See “Customizing Metadata At Runtime”.
9. Your EntityTable can offer properties not found in the actual table. They can reflect calculated values from other properties or any other data you’d like. See “Creating Calculated and Custom Columns”.

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

Creating the EntityTable Metadata Classes

The EntityTable Metadata Classes are companion classes to each EntityTable class. They have matching property names where the business logic attributes are applied. They are used when the EntityTable class may get overwritten as the database structure is modified.

- Use this section when you are using LINQ to SQL and ADO.NET Entity Framework. Their EntityTable classes are overwritten by a code generator, which demands that your business logic attributes are placed elsewhere.
- Skip this section when you have created the EntityTable classes by hand. For example, when using the Versatile DataSources with ADO.NET.

Before any business logic attributes can be applied, you need to:

- Create a code file where your classes will be added.
- Create partial classes for each EntityTable class
- Create new classes that parallel the EntityTable classes
- Use the MetadataTypeAttribute on the EntityTable class to link it to the EntityTable Metadata class.

The steps follow:

1. Add a code file to the **App_Code** folder.

   Naming suggestion: *your database name*Metadata (with .cs or .vb extension)

   When using Web Application Projects, set the property **Build** to `Compile` on the code file.

2. Open the code file in an editor.
3. Declare these namespaces at the top of your code file.

   [C#]

```
using System;
using System.Data;
using System.Linq;
using System.Xml.Linq;
using System.ComponentModel;
using System.ComponentModel.DataAnnotations;
using System.Runtime.Serialization;
using System.Collections.Generic;
using System.Web.DynamicData;
using DES=PeterBlum.DES.DynamicData;
```

   [VB]

```
Imports System
Imports System.Data
Imports System.Linq
Imports System.Xml.Linq
Imports System.ComponentModel
Imports System.ComponentModel.DataAnnotations
Imports System.Runtime.Serialization
Imports System.Collections.Generic
Imports System.Web.DynamicData
Imports DES=PeterBlum.DES.DynamicData
```

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

4. Create a class definition for each table name using the `Partial` keyword. It will have no properties or methods.

Example

[C#]

```
public partial class Category
{
}
```

[VB]

```
Partial Public Class Category
End Class
```

5. Create an EntityTable Metadata Class for each class added in step 4.

   Naming suggestion: *TableName*Metadata

Example

[C#]

```
public class CategoryMetadata
{
}
```

[VB]

```
Public Class CategoryMetadata
End Class
```

6. Use the MetadataTypeAttribute on the EntityTable classes to connect it to the EntityTable Metadata Classes.

[C#]

```
[MetadataType(typeof(CategoryMetadata))]
public partial class Category
{
}

public class CategoryMetadata
{
}
```

[VB]

```
<MetadataType(GetType(CategoryMetadata))> _
Partial Public Class Category
End Class

Public Class CategoryMetadata
End Class
```

Example files based on the NorthWind database and LINQ to SQL:

[C#]

[DES Product Folder]/DynamicData/CS/Samples/App_Code/NorthWindMetadata.cs

[VB]

[DES Product Folder]/DynamicData/VB/Samples/App_Code/NorthWindMetadata.vb

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

Applying Attributes To Columns

Attributes – classes derived from System.Attribute – allow you to annotate properties with metadata. Dynamic Data uses the metadata to set up each MetaColumn class according to your business logic.

Remember that there are now 3 classes representing each table:

- The original "EntityTable" that was generated for you by LINQ to SQL or ADO.NET Entity Framework
- The Partial class with the same name, in the code file you created
- The EntityTable Metadata Class, in the code file you created.

You will be applying attributes to the EntityTable Metadata Class only. Start by defining properties on that class, as described in the next topic.

Click on any of these topics to jump to them:

- Steps for defining your business logic

Business rule attributes:

- DataTypeAttributes: The DataTypes used by your Business logic
- ValidationAttributes: Applying Constraints on Columns
  - Writing server side only Validation Code within your EntityTable Classes
- DependencyAttributes: Relationships between columns
- UIHintAttribute: Customizing the Field Template used
- DisplayNameAttribute: Friendly column name
- DescriptionAttribute: A Hint for the user
- DefaultValueAttribute: Initial value when creating a new record
- DisplayFormatAttribute: Rules to convert from value to a string
- ScaffoldColumnAttribute: Determining if Scaffolding adds a Column
- SortableColumnAttribute: Determining sorting rules
- CategoryAttribute: Assigning Categories to Data Fields
- EditableAttribute: Prevent editing on certain columns
- ColumnRestrictionAttribute: Deny access to columns by Role
- CacheListAttribute: Caching Lists for Enumerated and ForeignKeys

- Customizing Metadata At Runtime
  - Understanding the ModifiableColumn class
  - Implementing the ICustomizeColumn Interface
  - Implementing the IUICustomizeColumn Interface

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

DataTypeAttributes: The DataTypes used by your Business logic

Each property has a type, such as string or int, that mimics the type from the Column in the table (nvarchar, integer). That type is often not specific enough for developing a user interface or validation rules. For example, a double can be used as a currency, percentage, duration, longitude, etc. DateTime can be used as a Date, Time, both or Duration. The **DataTypeAttribute** lets you refine the type so the right Field Template can be selected. Instead of just getting a textbox that validates decimals, it may get a CurrencyTextBox or PercentageTextBox with right validators.

Not every column (property in the EntityTable Metadata Class) needs a DataTypeAttribute. Use the DataTypeAttribute when the column:

- Must be described more precisely than the database's datatype.
- The Field Template provides a textbox to edit its value and the value is a datatype supported by the DataTypeCheckValidator. For example, when the datatype is a date, the editor is a DateTextBox which needs a DataTypeCheckValidator for DataType="Date". DES's smart DataTypeAttributes automatically install a correctly configured DataTypeCheckValidator.

See "Mapping DataType to its DataTypeAttribute".

As a rule, you should always apply the DataTypeCheckValidator to any textbox that contains a specialized datatype, like a number or date. If you provide the right DataTypeAttribute to the Column metadata, you are assured this will happen. In addition, DES Dynamic Data enforces this rule by creating the correct DataTypeCheckValidator for you on any column where you didn't set it up explicitly. For example, you have a column whose type is Integer and has no DataTypeAttribute. The **Integer_Edit.ascx** Field Template will still include a DataTypeCheckValidator to evaluate the IntegerTextBox.

Click on any of these topics to jump to them:

- Mapping DataType to its DataTypeAttribute
- DES.DataTypeAttribute
- DES.IntegerDataTypeAttribute
- DES.DecimalDataTypeAttribute
- DES.CurrencyDataTypeAttribute
- DES.PercentDataTypeAttribute
- DES.IntegerMeasurementDataTypeAttribute
- DES.DecimalMeasurementDataTypeAttribute
- DES.DateDataTypeAttribute
- DES.TimeOfDayDataTypeAttribute
- DES.DurationDataTypeAttribute
- DES.AnniversaryDataTypeAttribute
- DES.MonthYearDataTypeAttribute
- DES.EnumeratedAttribute
- DES.EnumeratedStringAttribute
- DES.EmailAddressDataTypeAttribute
- DES.UrlDataTypeAttribute
- DES.UrlAsImageDataTypeAttribute
- DES.DbImageDataTypeAttribute

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

Mapping DataType to its DataTypeAttribute

Use this chart to select the correct DataTypeAttribute.

Click on the Attribute name to jump to its description and properties.		
DataType	**Attribute**	**Field Template used**
String (not multiline text)	DES.DataTypeAttribute. Pass `DataType.Text`	Text
String with Multiline Text	DES.DataTypeAttribute. Pass `DataType.MultilineText`	MultilineText
Integer	DES.IntegerDataTypeAttribute	Integer
Decimal	DES.DecimalDataTypeAttribute	Decimal
Currency	DES.CurrencyDataTypeAttribute	Currency
Percent	DES.PercentDataTypeAttribute	Percent
Integer used as a measurement	DES.IntegerMeasurementDataTypeAttribute	Measurement
Decimal used as a measurement	DES.DecimalMeasurementDataTypeAttribute	Measurement
Date	DES.DateDataTypeAttribute	Date
Time of day	DES.TimeOfDayDataTypeAttribute	Time
Duration	DES.DurationDataTypeAttribute	Duration
DateTime	DES.DataTypeAttribute. Pass `DataType.DateTime`	DateTime
Date without year	DES.AnniversaryDataTypeAttribute	Anniversary
Date without day of month	DES.MonthYearDataTypeAttribute	MonthYear
Integer for an Enumerated type	DES.EnumeratedAttribute	Enumerated
String for an Enumerated type	DES.EnumeratedStringAttribute	Enumerated
Phonenumber	DES.DataTypeAttribute. Pass `DataType.PhoneNumber`	Text**
Password	DES.DataTypeAttribute. Pass `DataType.Password`	Password
Url	DES.UrlDataTypeAttribute	Url
Url to an image	DES.UrlAsImageDataTypeAttribute	UrlAsImage
Binary storing an image	DES.DbImageDataTypeAttribute	DbImage
Email address(es)	DES.EmailAddressDataTypeAttribute	EmailAddress
Anything else	DES.DataTypeAttribute. Pass a string with the datatype name.	*Matches the string passed.*

** If you create a Field Template, it will be automatically used only when the file name matches the DataType value.

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

DES.DataTypeAttribute

Use this when there no other DataTypeAttribute class applies. See the chart on the previous page.

There are actually two DataTypeAttribute classes, `System.ComponentModel.DataAnnotations.DataTypeAttribute` and `PeterBlum.DES.DynamicData.DataTypeAttribute`. While you can use either, generally its easiest to use DES's DataTypeAttribute because it supports all datatypes in the native attribute plus provides a few of its own.

There are three types you can pass to this attribute: `System.ComponentModel.DataAnnotations.DataType`, PeterBlum.DES.DynamicData.ExtendedDataType, and String. The following topics cover each.

Click on any of these topics to jump to them:

- System.ComponentModel.DataAnnotations.DataType
- PeterBlum.DES.DynamicData.ExtendedDataType
- String
- How to determine the DataType from within a Field Template

For other DataTypeAttributes, see "DataTypeAttributes: The DataTypes used by your Business logic".

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

System.ComponentModel.DataAnnotations.DataType

A list of types supplied with ASP.NET 3.5. This enumerated type has these values:

Value	Description
`Custom`	Used automatically when you pass a string to the DataTypeAttribute. You don't set it yourself.
`DateTime`	For DateTime values. This is automatically used when the property uses System.DateTime. The Field Templates used are DateTime.ascx, DateTime_Edit.ascx, and DateTime_Insert.ascx.
`Date`	For Date values. Use the **DES.DateDataTypeAttribute** if the Field Template uses the DateTextBox (which DES's Date_Edit.ascx does.)
`Time`	For Time of Day values. Use the **DES.TimeOfDayDataTypeAttribute** if the Field Template uses the TimeOfDayTextBox (which DES's Time_Edit.ascx does.)
`Duration`	For Duration values. Use the **DES.DurationDataTypeAttribute** if the Field Template uses the DurationTextBox (which DES's Duration_Edit.ascx does.)
`Currency`	For Currency values. Use the **DES.CurrencyDataTypeAttribute** if the Field Template uses the CurrencyTextBox (which DES's Currency_Edit.ascx does.)
`Text`	For string values. If your string supports multiline text or allows a large number of characters, choose MultilineText. You generally don't need to use DataType.Text because its automatically used for a string type.
`MultilineText`	For string values, especially when they have carriage returns or do not limit the number of characters to something small. It uses MultilineText_Edit.ascx for a Field Template when in edit mode. Dynamic Data generally omits this column when in readonly mode because the text can be excessive.
`Html`	For string values containing HTML.The Field Templates used are Text.ascx, Text_Edit.ascx, and Text_Insert.ascx. Text.ascx will apply HtmlEncoding, showing the HTML tags to the user. (You can override this by using a DynamicControl or DynamicField, both which have an HtmlEncode property that you can set to false.) Text_Edit.ascx provides a standard textbox. If you have a web control that edits HTML (such as a "rich textbox"), create a Field Template named Html_Edit.ascx to use that control by default.
`EmailAddress`	For strings that represent an email address. If you create a Field Template called "EmailAddress_Edit.ascx", it will automatically be used. Without it, the Field Template used is Text_Edit.ascx.
`Password`	For strings that represent passwords. The Field Template Password.ascx will not show any value for security reasons. The Field Template Password_Edit.ascx uses a textbox with TextMode=Password.
`PhoneNumber`	For strings that represent a phone number. If you create a Field Template called "PhoneNumber_Edit.ascx", it will automatically be used. Without it, the Field Template used is Text_Edit.ascx.
`Url`	For strings that represent a URL. If the Url is to an image, you may want to use the UrlAsImageDataTypeAttribute instead.

Example

[C#]

```
[DES.DataType(DataType.MultilineText)]
public object Description { get; set; }
```

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

[VB]

```
<DES.DataType(DataType.MultilineText)> _
Public Property Description As Object
  Get
    Return Nothing
  End Get
  Set
  End Set
End Property
```

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

PeterBlum.DES.DynamicData.ExtendedDataType

The native type `System.ComponentModel.DataAnnotations.DataType` is a fixed list of data types. In order for DES to offer more, they had to be created on a separate enumerated type, `PeterBlum.DES.DynamicData.ExtendedDataType`. Use this when one of the following datatypes are needed:

Value	Description
`Percent`	For Percent values. Use the **PercentDataTypeAttribute** if the Field Template uses the PercentTextBox (which DES's Percent_Edit.ascx does.) PercentDataTypeAttribute supports both integer and decimal column values.
`Measurement`	For integers and decimals that represent a measurement. While a measurement is really just a number, it also is connected to labels that describe the type of measurement. For example, "Gallon", "Decibels", and "Meters". Generally you use the **IntegerMeasurementDataTypeAttribute** or **DecimalMeasurementDataTypeAttribute** because they let you define those labels in singular, plural and abbreviated form. The Measurement.ascx and Measurement_Edit.ascx Field Templates automatically display these labels.
`MonthYear`	For dates were the day of month is not used. Use the **MonthYearDataTypeAttribute** if the Field Template uses the MonthYearTextBox (which DES's MonthYear_Edit.ascx does.)
`Anniversary`	For dates were the year is not used. Use the **AnniversaryDataTypeAttribute** if the Field Template uses the AnniversaryTextBox (which DES's Anniversary_Edit.ascx does.)
`CreditCardNumber`	For credit card numbers. DES will use the same Field Templates as with DataType.Text. If you create a specialized Field Template, the recommended Field Template names are CreditCardNumber.ascx, CreditCardNumber_Edit.ascx, and CreditCardNumber_Insert.ascx.
`ABARoutingNumber`	For ABA Routing numbers. DES will use the same Field Templates as with DataType.Text. If you create a specialized Field Template, the recommended Field Template names are ABARoutingNumber.ascx, ABARoutingNumber_Edit.ascx, and ABARoutingNumber_Insert.ascx.

Example

[C#]

```
[DES.DataType(DES.ExtendedDataType.MonthYear)]
public object ExpiryDate { get; set; }
```

[VB]

```
<DES.DataType(DES.ExtendedDataType.MonthYear)> _
Public Property ExpiryDate As Object
   Get
      Return Nothing
   End Get
   Set
   End Set
End Property
```

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

String

If you have a type not found in the above enumerated types, pass its name as a string to the DataTypeAttribute. It will use Field Templates with the same name.

Example

[C#]

```
[DES.DataType("NorthAmericanPhoneNumber")]
public object PhoneNumber { get; set; }
```

[VB]

```
<DES.DataType("NorthAmericanPhoneNumber")> _
Public Property PhoneNumber As Object
   Get
      Return Nothing
   End Get
   Set
   End Set
End Property
```

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

How to determine the DataType from within a Field Template

When creating a Field Template that supports multiple datatypes, use the MetaColumn to learn about the current DataTypeAttribute with its **DataTypeAttribute** property. (That property will be `null` if none is specified.) Then evaluate these properties found on `PeterBlum.DES.DynamicData.DataTypeAttribute`:

- DataType (enum System.ComponentModel.DataAnnotations.DataType) – Will be `DataType.Custom` when not used.
- ExtendedDataType (enum PeterBlum.DES.DynamicData.ExtendedDataType) – Will be `ExtendedDataType.NotUsed` when not used.
- CustomDataType (string) – Used by both String and ExtendedDataType.

Example

Suppose you create a Field Template that handles phone numbers. Your template sets up two patterns, North American and all others. All others would be `DataType.PhoneNumber` in this example.

[C#]

```
if (this.Column.DataTypeAttribute != null)      // this=Field Template class
  if (this.Column.DataTypeAttribute.DataType == DataType.Custom)
    switch (this.Column.DataTypeAttribute.CustomDataType)
    {
      case "NorthAmericanPhoneNumber":
        // format controls for (###) ### #### pattern
        break;
    }
  else
    switch (this.Column.DataTypeAttribute.DataType)
    {
      case DataType.PhoneNumber:
        // format controls for any phone number
        break;
    }
```

[VB]

```
If Not Me.Column.DataTypeAttribute Is Nothing Then    ' Me=Field Template class
  If Me.Column.DataTypeAttribute.DataType = DataType.Custom Then
    Select Case Me.Column.DataTypeAttribute.CustomDataType
      Case "NorthAmericanPhoneNumber"
        ' format controls for (###) ### #### pattern
    End Select
  Else
    Select Case Me.Column.DataTypeAttribute.DataType
      Case DataType.PhoneNumber
        ' format controls for any phone number
    End Select
  End If
End If
```

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

DES.IntegerDataTypeAttribute

Class	PeterBlum.DES.DynamicData.IntegerDataTypeAttribute
Supported Types	32 bit integer, 16 bit integer
Default Field Templates	Integer.ascx, Integer_Edit.ascx, and Integer_Insert.ascx
Validation	DataTypeCheckValidator with DataType="Integer"

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn Method". Be aware that you cannot create a DataTypeAttribute programmatically to establish a specific Field Template, but you can use it to effect a Validator on the Field Template.

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the DataTypeCheckValidator. Assigns these properties on that validator. You can also setup error message properties using the Customizer control's **ErrorMessages** property.
- **AllowNegatives** (boolean) – When `true`, both positive and negative values are allowed. When `false`, only positive values are allowed. It defaults to `true`.
- **Enabled** (boolean) – When `true`, the attribute is used. It defaults to `true`.

Example

[C#]

```
[DES.IntegerDataType(AllowNegatives=false)]
public object QuantityInStock { get; set; }
```

[VB]

```
<DES.IntegerDataType(AllowNegatives:=False)> _
Public Property QuantityInStock As Object
  Get
    Return Nothing
  End Get
  Set
  End Set
End Property
```

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

DES.DecimalDataTypeAttribute

Class	PeterBlum.DES.DynamicData.DecimalDataTypeAttribute
Supported Types	Double, Decimal
Default Field Templates	Decimal.ascx, Decimal_Edit.ascx, and Decimal_Insert.ascx
Validation	DataTypeCheckValidator with DataType="Decimal"

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn". Be aware that you cannot create a DataTypeAttribute programmatically to establish a specific Field Template, but you can use it to effect a Validator on the Field Template.

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the DataTypeCheckValidator. Assigns these properties on that validator. You can also setup error message properties using the Customizer control's **ErrorMessages** property.
- **AllowNegatives** (boolean) – When `true`, both positive and negative values are allowed. When `false`, only positive values are allowed. It defaults to `true`.
- **MaxDecimalPlaces** (integer) – Determines the maximum number of decimal places allowed. When the user enters more, it is an error. Set to 0 to ignore this property. It defaults to 0.
- **AcceptPeriodAsDecimalSeparator** (boolean) – Many cultures do not use the period as the decimal separator. It makes numeric entry from the numeric keypad more difficult, because it features a period key. When `true`, both the culture's decimal separator and a period are allowed as the decimal separator. It defaults to `false`.
- **Enabled** (boolean) – When `true`, the attribute is used. It defaults to `true`.

Example

[C#]

```
[DES.DecimalDataType(MaxDecimalPlaces=2, AllowNegatives=false)]
public object Width { get; set; }
```

[VB]

```
<DES.DecimalDataType(MaxDecimalPlaces:=2, AllowNegatives:=False)> _
Public Property Width As Object
   Get
      Return Nothing
   End Get
   Set
   End Set
End Property
```

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

DES.CurrencyDataTypeAttribute

Class	PeterBlum.DES.DynamicData.CurrencyDataTypeAttribute
Supported Types	Double, Decimal
Default Field Templates	Currency.ascx, Currency_Edit.ascx, and Currency_Insert.ascx
Validation	DataTypeCheckValidator with DataType="Currency"

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn". Be aware that you cannot create a DataTypeAttribute programmatically to establish a specific Field Template, but you can use it to effect a Validator on the Field Template.

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the DataTypeCheckValidator. Assigns these properties on that validator. You can also setup error message properties using the Customizer control's **ErrorMessages** property.
- **AllowNegatives** (boolean) – When `true`, both positive and negative values are allowed. When `false`, only positive values are allowed. It defaults to `true`.
- **UseCurrencySymbol** (boolean) – When true, Field Templates can display the currency symbol. Currency.ascx will embed it into the label along with the number, following CultureInfo rules. Currency_Edit.ascx will setup the CurrencyTextBox to allow entry of the currency symbol. It defaults to `false`.
- **UseDecimalDigits** (boolean) – Determines if there is a limit for the number of decimal digits following **CultureInfo.NumberFormat.CurrentDecimalDigits** or the user is allowed to enter more digits. When `true`, the limit is imposed. When `false`, there is no limit. It defaults to `true`.
- **Enabled** (boolean) – When `true`, the attribute is used. It defaults to `true`.

Example

[C#]

```
[DES.CurrencyDataType(UseCurrencySymbol=false, AllowNegatives=false)]
public object RetailPrice { get; set; }
```

[VB]

```
<DES.CurrencyDataType(UseCurrencySymbol:=False, AllowNegatives:=False)> _
Public Property RetailPrice
   Get
      Return Nothing
   End Get
   Set
   End Set
End Property
```

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

DES.PercentDataTypeAttribute

Class	PeterBlum.DES.DynamicData.PercentDataTypeAttribute
Supported Types	Double, Decimal, 16 bit Integer, 32 bit Integer
Default Field Templates	Percent.ascx, Percent_Edit.ascx, and Percent_Insert.ascx
Validation	DataTypeCheckValidator with DataType="Percent"

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn". Be aware that you cannot create a DataTypeAttribute programmatically to establish a specific Field Template, but you can use it to effect a Validator on the Field Template.

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the DataTypeCheckValidator. Assigns these properties on that validator. You can also setup error message properties using the Customizer control's **ErrorMessages** property.
- **AllowNegatives** (boolean) – When `true`, both positive and negative values are allowed. When `false`, only positive values are allowed. It defaults to `true`.
- **UsePercentSymbol** (boolean) – When `true`, Field Templates can display the currency symbol. Percent.ascx will embed it into the label along with the number, following CultureInfo rules. Percent_Edit.ascx will setup the PercentTextBox to allow entry of the percent symbol. It defaults to `false`.
- **WholeNumbersOnly** (boolean) - When `true`, conversion to string will strip off the decimal separator and decimal digits. It defaults to `false`.
- **Enabled** (boolean) – When `true`, the attribute is used. It defaults to `true`.

Example

[C#]

```
[DES.PercentDataType(WholeNumbersOnly=true)]
public object Percent { get; set; }
```

[VB]

```
<DES.PercentDataType(WholeNumbersOnly:=True)> _
Public Property Percent As Object
   Get
      Return Nothing
   End Get
   Set
   End Set
End Property
```

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

DES.IntegerMeasurementDataTypeAttribute

A measurement is a numeric column that reflects a measurement, like gallon, millimeter, and degrees. This type lets you define labels that describe the measurement and can appear on the web form.

Class	PeterBlum.DES.DynamicData.IntegerMeasurementDataTypeAttribute
Supported Types	32 bit integer, 16 bit integer
Default Field Templates	Measurement.ascx, Measurement_Edit.ascx, and Measurement_Insert.ascx
Validation	DataTypeCheckValidator with DataType="Integer"

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn". Be aware that you cannot create a DataTypeAttribute programmatically to establish a specific Field Template, but you can use it to effect a Validator on the Field Template.

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the DataTypeCheckValidator. Assigns these properties on that validator. You can also setup error message properties using the Customizer control's **ErrorMessages** property.
- **Name** (string) – The measurement name in singular form. Use **PluralName** for plural form. Shown in validation error messages when using the token "{NAME}". It defaults to "`unit`".
- **NameLookupID** (string) – Uses the String Lookup System to determine the name in singular form. Looks for this value in the ValidationMisc string group. It defaults to "".
- **PluralName** (string) – The measurement name in plural form. Shown in validation error messages when using the token "{PLURALNAME}". It defaults to "`units`".
- **PluralNameLookupID** (string) – Uses the String Lookup System to determine the name in plural form. Looks for this value in the ValidationMisc string group. It defaults to "".
- **AbbrevName** (string) – The measurement name in abbreviated form. Shown in validation error messages when using the token "{ABBREVNAME}". It defaults to "".
- **AbbrevNameLookupID** (string) – Uses the String Lookup System to determine the name in abbreviated form. Looks for this value in the ValidationMisc string group. It defaults to "".
- **DataFormatString** (string) – Determines how to place the number and its name in the string shown with the Measurement.ascx Field Template.

  Uses these tokens: {VALUE} for the number, {NAME} for Name, {PLURALNAME} for PluralName, and {ABBREVNAME} for AbbrevName.

  It defaults to "{VALUE} {PLURALNAME}".
- **DataFormatStringLookupID** (string) – Uses the String Lookup System to determine the DataFormatString. Looks for this value in the ValidationMisc string group. It defaults to "".
- **HtmlEncode** (boolean) – Determines if HTMLEncoding should be applied to any resulting string from this attribute, including **Name**, **PluralName**, **AbbrevName** and **DataFormatString**. It defaults to `true`.
- **AllowNegatives** (boolean) – When `true`, both positive and negative values are allowed. When `false`, only positive values are allowed. It defaults to `false`.
- **Enabled** (boolean) – When `true`, the attribute is used. It defaults to `true`.

Example

[C#]

```
[DES.IntegerMeasurementDataType(Name="Liter", PluralName="Liters", AbbrevName="Lt")]
public object Quantity { get; set; }
```

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

[VB]

```
<DES.IntegerMeasurementDataType(Name:="Liter", PluralName:="Liters", _
  AbbrevName:="Lt")> _
Public Property Quantity As Object
  Get
    Return Nothing
  End Get
  Set
  End Set
End Property
```

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

DES.DecimalMeasurementDataTypeAttribute

A measurement is a numeric column that reflects a measurement, like gallon, millimeter, and degrees. This type lets you define labels that describe the measurement and can appear on the web form.

Class	PeterBlum.DES.DynamicData.DecimalMeasurementDataTypeAttribute
Supported Types	Double, Decimal
Default Field Templates	Measurement.ascx, Measurement_Edit.ascx, and Measurement_Insert.ascx
Validation	DataTypeCheckValidator with DataType="Decimal"

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn". Be aware that you cannot create a DataTypeAttribute programmatically to establish a specific Field Template, but you can use it to effect a Validator on the Field Template.

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the DataTypeCheckValidator. Assigns these properties on that validator. You can also setup error message properties using the Customizer control's **ErrorMessages** property.
- **Name** (string) – The measurement name in singular form. Use **PluralName** for plural form. Shown in validation error messages when using the token "{NAME}". It defaults to "`unit`".
- **NameLookupID** (string) – Uses the String Lookup System to determine the name in singular form. Looks for this value in the ValidationMisc string group. It defaults to "".
- **PluralName** (string) – The measurement name in plural form. Shown in validation error messages when using the token "{PLURALNAME}". It defaults to "`units`".
- **PluralNameLookupID** (string) – Uses the String Lookup System to determine the name in plural form. Looks for this value in the ValidationMisc string group. It defaults to "".
- **AbbrevName** (string) – The measurement name in abbreviated form. Shown in validation error messages when using the token "{ABBREVNAME}". It defaults to "".
- **AbbrevNameLookupID** (string) – Uses the String Lookup System to determine the name in abbreviated form. Looks for this value in the ValidationMisc string group. It defaults to "".
- **DataFormatString** (string) – Determines how to place the number and its name in the string shown with the Measurement.ascx Field Template.

  Uses these tokens: {VALUE} for the number, {NAME} for Name, {PLURALNAME} for PluralName, and {ABBREVNAME} for AbbrevName.

  It defaults to "{VALUE} {PLURALNAME}".
- **DataFormatStringLookupID** (string) – Uses the String Lookup System to determine the DataFormatString. Looks for this value in the ValidationMisc string group. It defaults to "".
- **HtmlEncode** (boolean) – Determines if HTMLEncoding should be applied to any resulting string from this attribute, including **Name**, **PluralName**, **AbbrevName** and **DataFormatString**. It defaults to `true`.
- **AllowNegatives** (boolean) – When `true`, both positive and negative values are allowed. When `false`, only positive values are allowed. It defaults to `false`.
- **Enabled** (boolean) – When `true`, the attribute is used. It defaults to `true`.

Example

[C#]

```
[DES.DecimalMeasurementDataType(Name="Liter", PluralName="Liters", AbbrevName="Lt")]
public object Quantity { get; set; }
```

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

[VB]

```
<DES.DecimalMeasurementDataType(Name:="Liter", PluralName:="Liters", _
  AbbrevName:="Lt")> _
Public Property Quantity As Object
  Get
    Return Nothing
  End Get
  Set
  End Set
End Property
```

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

DES.DateDataTypeAttribute

Class	PeterBlum.DES.DynamicData.DateDataTypeAttribute
Supported Types	DateTime
Default Field Templates	Date.ascx, Date_Edit.ascx, and Date_Insert.ascx
Validation	DataTypeCheckValidator with DataType="Date"

Note: If you want a DateTime that uses a Field Template supporting both date and time, use DataTypeAttribute(System.ComponentModel.DataType.DateTime). It will use these Field Templates: DateTime.ascx, DateTime_Edit.ascx, and DateTime_Insert.ascx.

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn". Be aware that you cannot create a DataTypeAttribute programmatically to establish a specific Field Template, but you can use it to effect a Validator on the Field Template.

Note: While there are no properties here to adjust the formatting. The Date Field Templates have these properties which you can set on the UIHintAttribute, DynamicField and DynamicColumn: DateFormat, AllowMonthNames

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the DataTypeCheckValidator. Assigns these properties on that validator. You can also setup error message properties using the Customizer control's **ErrorMessages** property.
- **Enabled** (boolean) – When `true`, the attribute is used. It defaults to `true`.

Example

[C#]

```
[DES.DateDataType()]
public object BirthDate { get; set; }
```

[VB]

```
<DES.DateDataType()> _
Public Property BirthDate As Object
  Get
    Return Nothing
  End Get
  Set
  End Set
End Property
```

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

DES.TimeOfDayDataTypeAttribute

Class	PeterBlum.DES.DynamicData.TimeOfDayDataTypeAttribute
Supported Types	DateTime, TimeSpan
Default Field Templates	Time.ascx, Time_Edit.ascx, and Time_Insert.ascx
Validation	DataTypeCheckValidator with DataType="TimeOfDay"

Note: If you want a DateTime that uses a Field Template supporting both date and time, use DataTypeAttribute(System.ComponentModel.DataType.DateTime). It will use these Field Templates: DateTime.ascx, DateTime_Edit.ascx, and DateTime_Insert.ascx.

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn". Be aware that you cannot create a DataTypeAttribute programmatically to establish a specific Field Template, but you can use it to effect a Validator on the Field Template.

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the DataTypeCheckValidator. Assigns these properties on that validator. You can also setup error message properties using the Customizer control's **ErrorMessages** property.
- **TimeFormat** (enum PeterBlum.DES.TimeFormat) – Determines whether the time shows in 12 hour or 24 hour format. When 12 hour format, it uses the "AM" and "PM" designators, defined in **CultureInfo.DateTimeFormat.AMDesignator** and **CultureInfo.DateTimeFormat.PMDesignator**.

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

  - `Auto` – Uses the case from the letter H in the **CultureInfo.DateTimeFormat.LongTimePattern** to determine 12 or 24 hours. When it is lowercase ("h:mm:ss"), it is 12 hours. When it is uppercase ("H:mm:ss"), it is 24 hours. (This is a standard used by .net in the LongTimePattern.) This is the default.
  - `Hr12` – Use 12 hour format.
  - `Hr24` – Use 24 hour format.
- **ShowSeconds** (enum PeterBlum.DES.BaseTimeTextBox.ShowSecondsMode) – Determines how seconds are shown.

  The enumerated type `PeterBlum.DES.BaseTimeTextBox.ShowSeconds` has these values:

  - `No` – Seconds are never shown. If the user enters seconds, they will be stripped upon field validation.
  - `Yes` – Seconds are always shown.
  - `NonZero` – Seconds are shown when they are non-zero. They are hidden when zero. If the user enters 0 for seconds, they will be stripped upon field validation. This is the default.
- **DecimalMode** (enum PeterBlum.DES.BaseTimeTextBox.TimeDecimalMode) – Time entry optionally permits the period character to be used in these ways:
  - Allow decimal numbers for hours and minutes (1.5 = 1:30)
  - It is an alternative for the timeseparator (1.30 = 1:30).
  - It can also be turned off.

  The enumerated type `PeterBlum.DES.BaseTimeTextBox.TimeDecimalMode` has these values:

  - `NotUsed` – Do not permit the decimal separator.
  - `Decimal` – Detect a decimal value and convert it into hours and minutes. This is the default.
  - `TimeSeparator` – The decimal is the same as a time separator character, representing hours(period)minutes. The time separator is defined on **DateTimeFormatInfo.TimeSeparator**.
- **Enabled** (boolean) – When `true`, the attribute is used. It defaults to `true`.

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

Example

[C#]

```
[DES.TimeOfDayDataType(TimeFormat=PeterBlum.DES.TimeFormat.Hr24)]
public object OpeningTime { get; set; }
```

[VB]

```
<DES.IntegerDataType(TimeFormat:=PeterBlum.DES.TimeFormat.Hr24)> _
Public Property OpeningTime As Object
  Get
    Return Nothing
  End Get
  Set
  End Set
End Property
```

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

DES.DurationDataTypeAttribute

Class	PeterBlum.DES.DynamicData.DurationDataTypeAttribute
Supported Types	DateTime, TimeSpan, Integer, Decimal, and Double. When using integer or decimal, use the Conversion property to indicate if the value is in milliseconds, seconds, minutes, hours, or days.
Default Field Templates	Duration.ascx, Duration_Edit.ascx, and Duration_Insert.ascx
Validation	DataTypeCheckValidator with DataType="Duration"

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn". Be aware that you cannot create a DataTypeAttribute programmatically to establish a specific Field Template, but you can use it to effect a Validator on the Field Template.

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the DataTypeCheckValidator. Assigns these properties on that validator. You can also setup error message properties using the Customizer control's **ErrorMessages** property.
- **Conversion** (enum PeterBlum.DES.DynamicData.DurationConversion) – When the property type is integer or decimal, this property determines what is really stored as: milliseconds, seconds, minutes, hours or days.

  This value is used by the Duration.ascx and Duration_Edit.ascx Field Templates.

  The enumerated type `PeterBlum.DES.DynamicData.DurationConversion` has these values:

  - `Default` – Looks at the datatype. If its DateTime or TimeSpan, its time is used. If decimal, it is treated as hours. If integer, it is treated as seconds. This is the default.
  - `AsMilliseconds`
  - `AsSeconds`
  - `AsMinutes`
  - `AsHours`
  - `AsDays`
- **ShowSeconds** (enum PeterBlum.DES.BaseTimeTextBox.ShowSecondsMode) – Determines how seconds are shown.

  The enumerated type `PeterBlum.DES.BaseTimeTextBox.ShowSeconds` has these values:

  - `No` – Seconds are never shown. If the user enters seconds, they will be stripped upon field validation.
  - `Yes` – Seconds are always shown.
  - `NonZero` – Seconds are shown when they are non-zero. They are hidden when zero. If the user enters 0 for seconds, they will be stripped upon field validation. This is the default.
- **DecimalMode** (enum PeterBlum.DES.BaseTimeTextBox.TimeDecimalMode) – Time entry optionally permits the period character to be used in these ways:
  - Allow decimal numbers for hours and minutes (1.5 = 1:30)
  - It is an alternative for the timeseparator (1.30 = 1:30).
  - It can also be turned off.

  The enumerated type `PeterBlum.DES.BaseTimeTextBox.TimeDecimalMode` has these values:

  - `NotUsed` – Do not permit the decimal separator.
  - `Decimal` – Detect a decimal value and convert it into hours and minutes. This is the default.
  - `TimeSeparator` – The decimal is the same as a time separator character, representing hours(period)minutes. The time separator is defined on **DateTimeFormatInfo.TimeSeparator**.

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

- **Enabled** (boolean) – When `true`, the attribute is used. It defaults to `true`.

Example

[C#]

```
[DES.DurationDataType(Conversion=DES.DurationConversion.AsSeconds)]
public object FinishTime { get; set; }
```

[VB]

```
<DES.DurationDataType(Conversion:=DES.DurationConversion.AsSeconds)> _
Public Property FinishTime As Object
  Get
    Return Nothing
  End Get
  Set
  End Set
End Property
```

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

DES.AnniversaryDataTypeAttribute

Class	PeterBlum.DES.DynamicData.AnniversaryDataTypeAttribute
Supported Types	DateTime
Default Field Templates	Anniversary.ascx, Anniversary_Edit.ascx, and Anniversary_Insert.ascx
Validation	DataTypeCheckValidator with DataType="Anniversary"

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn". Be aware that you cannot create a DataTypeAttribute programmatically to establish a specific Field Template, but you can use it to effect a Validator on the Field Template.

Note: While there are no properties here to adjust the formatting. The Anniversary Field Templates have these properties which you can set on the DynamicField and DynamicColumn: DateFormat, AllowMonthNames

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the DataTypeCheckValidator. Assigns these properties on that validator. You can also setup error message properties using the Customizer control's **ErrorMessages** property.
- **Enabled** (boolean) – When `true`, the attribute is used. It defaults to `true`.

Example

[C#]

```
[DES.AnniversayDataType()]
public object BirthDay { get; set; }
```

[VB]

```
<DES.AnniversaryDataType()> _
Public Property BirthDay As Object
  Get
    Return Nothing
  End Get
  Set
  End Set
End Property
```

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

DES.MonthYearDataTypeAttribute

Class	PeterBlum.DES.DynamicData.MonthYearDataTypeAttribute
Supported Types	DateTime
Default Field Templates	MonthYear.ascx, MonthYear_Edit.ascx, and MonthYear_Insert.ascx
Validation	DataTypeCheckValidator with DataType="MonthYear"

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn". Be aware that you cannot create a DataTypeAttribute programmatically to establish a specific Field Template, but you can use it to effect a Validator on the Field Template.

Note: While there are no properties here to adjust the formatting. The Date Field Templates have these properties which you can set on the DynamicField and DynamicColumn: DateFormat, AllowMonthNames

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the DataTypeCheckValidator. Assigns these properties on that validator. You can also setup error message properties using the Customizer control's **ErrorMessages** property.
- **Enabled** (boolean) – When `true`, the attribute is used. It defaults to `true`.

Example

[C#]

```
[DES.MonthYearDataType()]
public object Expiry { get; set; }
```

[VB]

```
<DES.MonthYearDataType()> _
Public Property Expiry As Object
  Get
    Return Nothing
  End Get
  Set
  End Set
End Property
```

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

DES.EnumeratedAttribute

For integer columns that have a fixed list of values and associated text for each value, much like an enumerated type. In fact, you can pass an enumerated type to this attribute and it will learn the correct mapping of number to text. You can also create other mappings and support localization. Field Templates generally show a list or DropDownList with the textual values.

Note: ASP.NET 4 offers the System.ComponentModel.DataAnnotations.EnumDataTypeAttribute. It is supported by DES's Field Templates. However, the DES.EnumeratedAttribute and DES.EnumeratedStringAttribute are far more flexible and powerful, allowing you to show user friendly names instead of names from your code.

Class	PeterBlum.DES.DynamicData.EnumeratedAttribute
Supported Types	32 bit integer, 16 bit integer
Default Field Templates	Enumerated.ascx, Enumerated_Edit.ascx, and Enumerated_Insert.ascx
Validation	none

Click on any of these topics to jump to them:

- Using this Attribute
- Properties
- Using the CustomizeColumn Method

Using this Attribute

An enumerated type is an integer column with a specific list of values. Often you show user-friendly names for those numbers, the same way an enumerated type is declared when programming.

You will use the EnumeratedAttribute to define each number and optionally map it to a name.

It gets the list of integer values and associated names from several sources:

- An enumerated type, which already has both name and integer values. However, the names are not necessarily user friendly as they cannot have spaces between words. Assign the type of your enumerated type to the **EnumeratedType** property.
- A string that maps values to names, defined in the **ValueNameList** property. It is a delimited string, using the pipe character as the default delimiter. Names and values are separated by another delimiter which defaults to "=".

  Format example: "Value1=Name1|Value2=Name2"
- The String Lookup System, which uses a base string in the **LookupIDLeadText** property and adds the value to form the complete lookupID. Format example: "BaseNameValue1". These strings go into the Misc string lookup group.
- User can programmatically add items in the `CustomizeColumn()` method on an EntityTable class that implements ICustomizeColumn.

Names can be supplied multiple times for a value. Each one overwrites the earlier one using the order shown above. As a result, each feature can contain a subset of the total strings. For example, to fix a single entry in the enumerated type, set the **EnumeratedType** property. Then set the **ValueNameList** property with just the value and its replacement name.

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

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn Method". Be aware that you cannot create a DataTypeAttribute programmatically to establish a specific Field Template.

- **EnumeratedType** (Type) – A source of strings from an enumerated type. It requires a real enumerated type. .net enumerated types always have integer values internally defined against the values you see in code. Unless you override it, elements of the enumerated type start at 0 and increment.

  The names are the text of each enumerated element.

  This type is the first to be converted into the final list. As a result, all other ways to assign the list will merge with these values. If they have the same value as an element added here, they replace the name for that value. Otherwise, they add.

  The list is ordered based on the elements of this type.

  It defaults to `null`.

  Example

  [C#]

  ```
  public enum PrimaryColors
  {
     red,
     yellow,
     blue
  }

  [DES.Enumerated(EnumeratedType=typeof(PrimaryColors)]
  ```

  [VB]

  ```
  Public Enum PrimaryColors
     red,
     yellow,
     blue
  End Enum

  <DES.Enumerated(EnumeratedType:=GetType(PrimaryColors)> _
  ```

  This results in the following used by a ListBox or DropDownList

0	"red"
1	"yellow"
2	"blue"

- **ValueNameList** (string) – A delimited list of values and names. The list must be in this pattern:

  ```
  Value1[separator]Name1[listdelimiter]Value2[separator]Name2
  ```

  Value elements are usually an integer. But if you use **EnumeratedType**, you can supply the textual value of the element instead of its integer value.

  The `[listdelimiter]` defaults to the pipe character but can be changed with the **ListDelimiter** property. The `[separator]` defaults to the equals character but can be changed with the **ValueNameSeparator**.

  This property is the second to be converted into the final list, after **EnumeratedType**.

  It will overwrite the names of already assigned values and add the rest to the end. When the name is "!!!", it will delete a value already added from **EnumeratedType**. For example, if you want to remove the first item from the enumerated type, use `ValueNameList="0=!!!"`.

  The list is ordered based on the elements of this type.

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

Example

[C#]

```
[DES.Enumerated(ValueNameList="0=red|1=yellow|2=blue"]
```

[VB]

```
<DES.Enumerated(ValueNameList:="0=red|1=yellow|2=blue"> _
```

Example: Rename values from an Enumerated Type

This uses the enumerated type shown in the **EnumeratedType** property. The user wants each name capitalized. Because the enumerated type is used, the value elements can be the text for each element instead of the integer value.

[C#]

```
[DES.Enumerated(EnumeratedType=typeof(PrimaryColors),
 ValueNameList="red=Red|yellow=Yellow|blue=Blue"]
```

[VB]

```
<DES.Enumerated(EnumeratedType:=GetType(PrimaryColors),
 ValueNameList:="red=Red|yellow=Yellow|blue=Blue"> _
```

0	“Red”
1	“Yellow”
2	“Blue”

Example: Expanding an EnumeratedType

This uses the enumerated type shown in the **EnumeratedType** property. The user wants two more values, 10 = white and 11 = black.

[C#]

```
[DES.Enumerated(EnumeratedType=typeof(PrimaryColors),
 ValueNameList="10=white|11=black"]
```

[VB]

```
<DES.Enumerated(EnumeratedType:=GetType(PrimaryColors),
 ValueNameList:="10=white|11=black"> _
```

0	“red”
1	“yellow”
2	“blue”
10	“white”
11	“black”

- **ListDelimiter** (char) – The delimiter between each name/value pair in **ValueNameList**.

  It defaults to the pipe character ("|").

- **ValueNameSeparator** (char) – The separator between each value and its name in the **ValueNameList** property.

  It defaults to the equals character ("=").

- **LookupIDLeadText** (string) – Uses the String Lookup System to define names. Each value already added to the list will be reviewed against the string group Misc to see if there is text to use there. As a result, you must setup an initial list using the **EnumeratedType** or **ValueNameList** properties.

  This property’s value is the first characters of the LookupID passed to the String Lookup System. It is followed by the value. For example, if the value is "2" and this is "MyID", the Misc table must have an entry of "MyID2" to get new text for the name used here.

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

When using **EnumeratedType**, you can supply the textual value of the element instead of its integer value. For example, when using enumerated type `HorizontalAlign`, **LookupIDLeadText**="`HorizAlign.`" and the enumerated element is "`Left`", the LookupID becomes "`HorizAlign.Left`".

Example

Suppose the Misc string group has this text:

Key	Value
HorizAlign.NotSet	!!!
HorizAlign.Left	To the left
HorizAlign.Center	Centered
HorizAlign.Right	To the right

Note: The value "!!!" will delete an entry for NotSet.

[C#]

```
[DES.Enumerated(EnumeratedType=typeof(HorizontalAlign),
 LookupIDLeadText="HorizAlign."]
```

[VB]

```
<DES.Enumerated(EnumeratedType:=GetType(HorizontalAlign), _
 LookupIDLeadText:="HorizAlign."> _
```

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

Using the CustomizeColumn Method

You can programmatically define or edit the list of values on an EnumeratedAttribute by implementing the `PeterBlum.DES.DynamicData.ICustomizeColumn` interface on your EntityTable or the `PeterBlum.DES.DynamicData.IUICustomizeColumn` interface on your web form. See "Customizing Metadata At Runtime". Both define the `CustomizeColumn()` method.

The `CustomizeColumn()` method is passed a `PeterBlum.DES.DynamicData.ModifiableColumn` object which contains methods to modify an existing EnumeratedAttribute. Make sure the column already has the EnumeratedAttribute assigned. When `CustomizeColumn()` is called, the EnumeratedAttribute will already be populated with a list based on what was defined in the metadata.

Method: ModifyEnumeratedList

[C#]

```
public void ModifyEnumeratedList(int pValue, string pName, bool pApplyStringLookup)
```

[VB]

```
Public Sub ModifyEnumeratedList(ByVal pValue As Integer, _
      ByVal pName As String, ByVal pApplyStringLookup As String)
```

Parameters

pValue

The integer value that will be mapped to the name. If the value already exists, it will be overwritten. Otherwise its added to the end of the list.

pName

The name shown to the user. If you pass "!!!", it will delete an entry with *pValue* that was previously defined.

pApplyStringLookup

When `true`, use the String Lookup System to modify the value in *pName*. If the **LookupIDLeadText** property is defined on the EnumeratedAttribute and a key can be found in the String Lookup System, the value returned from the String Lookup System is used (except if blank). Otherwise the original value of *pName* is used.

Example

Populate the Region field of the Customer record with country names.

[C#]

```
[MetadataType(typeof(CustomerMetadata))]
public partial class Customer : DES.ICustomizeColumn
{
   public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn, bool pIsNewRow)
   {
      switch (pModifiableColumn.ColumnName)
      {
         case "Region":
            pModifiableColumn.ModifyEnumeratedList(1, "United States", false);
            pModifiableColumn.ModifyEnumeratedList(2, "Canada", false);
            pModifiableColumn.ModifyEnumeratedList(3, "Mexico", false);
            break;
      }  // switch
   }
}
public class CustomerMetadata
{
   [DES.Enumerated()]
   public object Region {get; set; }
}
```

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

[VB]

```
<MetadataType(GetType(CustomerMetadata))> _
Partial Public Class Customer _
     Implements DES.ICustomizeColumn

   Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn,_
     ByVal pIsNewRow As Boolean)

      Select Case pModifiableColumn.ColumnName
         Case "Region"
            pModifiableColumn.ModifyEnumeratedList(1, "United States", False)
            pModifiableColumn.ModifyEnumeratedList(2, "Canada", False)
            pModifiableColumn.ModifyEnumeratedList(3, "Mexico", False)
      End Select
   End Sub
End Class
Public Class CustomerMetadata
  <DES.Enumerated() > _
  Public Property Region As Object
     Get
       Return Nothing
     End Get
     Set
     End Set
  End Property
End Class
```

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

DES.EnumeratedStringAttribute

Note: ASP.NET 4 offers the System.ComponentModel.DataAnnotations.EnumDataTypeAttribute. It is supported by DES's Field Templates. However, the DES.EnumeratedAttribute and DES.EnumeratedStringAttribute are far more flexible and powerful, allowing you to show user friendly names instead of names from your code.

Class	PeterBlum.DES.DynamicData.EnumeratedStringAttribute
Supported Types	string
Default Field Templates	Enumerated.ascx, Enumerated_Edit.ascx, and Enumerated_Insert.ascx
Validation	none

Click on any of these topics to jump to them:

- Using this Attribute
- Properties
- Using the CustomizeColumn Method

Using this Attribute

This is a variation of DES.EnumeratedAttribute, designed for columns that contain strings. Use it to limit the strings offered in a field using a list. You can supply alternative text and localized text.

You will use the EnumeratedStringAttribute to define each string stored in your database and optionally map it to alternative text shown to the user name. The text shown to the user is the "name". If not optionally mapped, the original string from the database is used as the name.

It gets the list of string values and optional names from several sources:

- An enumerated type, assigned to the **EnumeratedType** property. The strings for values and names are taken from one source, the text value of each element in the enumerated type.
- A delimited string that lists the string values and optionally maps to alternative text. This is defined in the **ValueNameList** property. It is a delimited string, using the pipe character as the default delimiter. Names and values are separated by another delimiter which defaults to "=".

  Format example: "Value1|Value2|Value3=Name3|Value4=Name4"

  where Value1 and Value2 do not get mapped to alternative text.
- The String Lookup System, which uses a base string in the **LookupIDLeadText** property and adds the value to form the complete lookupID. Format example: "BaseNameValue1". These strings go into the Misc string lookup group.
- User can programmatically add items in the `CustomizeColumn()` method on an EntityTable class that implements ICustomizeColumn.

Names can be supplied multiple times for a value. Each one overwrites the earlier one using the order shown above. As a result, each feature can contain a subset of the total strings. For example, to fix a single entry in the enumerated type, set the **EnumeratedType** property. Then set the **ValueNameList** property with just the value and its replacement name.

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

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the ". Be aware that you cannot create a DataTypeAttribute programmatically to establish a specific Field Template.

- **EnumeratedType** (Type) – A source of strings from an enumerated type. It requires a real enumerated type. It creates a list of strings using the text of each element in the enumerated type.

  This type is the first to be converted into the final list. As a result, all other ways to assign the list will merge with these values. If they have the same value as an element added here, they replace the name for that value. Otherwise, they add.

  The list is ordered based on the elements of this type.

  It defaults to `null`.

  Example

  [C#]

  ```
  public enum PrimaryColors
  {
     red,
     yellow,
     blue
  }
  [DES.EnumeratedString(EnumeratedType=typeof(PrimaryColors)]
  ```

  [VB]

  ```
  Public Enum PrimaryColors
     red,
     yellow,
     blue
  End Enum
  <DES.EnumeratedString(EnumeratedType:=GetType(PrimaryColors)> _
  ```

  This results in the following used by a ListBox or DropDownList

Value	Name
"red"	"red"
"yellow"	"yellow"
"blue"	"blue"

- **ValueNameList** (string) – A delimited list of values and optionally alternative names. The list must be in this pattern:

  ```
  Value1[listdelimiter]Value2[listdelimiter]Value3
  ```

  When specifying an alternative name, use Value1[separator]Name1 like this:

  ```
  Value1[separator]Name1[listdelimiter]Value2[separator]Name2
  ```

  The `[listdelimiter]` defaults to the pipe character but can be changed with the **ListDelimiter** property. The `[separator]` defaults to the equals character but can be changed with the **ValueNameSeparator**.

  This property is the second to be converted into the final list, after **EnumeratedType**.

  It will overwrite the names of already assigned values and add the rest to the end. When the name is "!!!", it will delete a value already added from **EnumeratedType**. For example, if you want to remove the first item from the enumerated type, use `ValueNameList="0=!!!"`.

  The list is ordered based on the elements of this type.

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

Example

[C#]

```
[DES.Enumerated(ValueNameList="red|yellow|blue"]
```

[VB]

```
<DES.Enumerated(ValueNameList:="red|yellow|blue"> _
```

Example: Rename values from an Enumerated Type

This uses the enumerated type shown in the **EnumeratedType** property. The user wants each name capitalized. Because the enumerated type is used, the value elements can be the text for each element instead of the integer value.

[C#]

```
[DES.Enumerated(EnumeratedType=typeof(PrimaryColors),
 ValueNameList="red=Red|yellow=Yellow|blue=Blue"]
```

[VB]

```
<DES.Enumerated(EnumeratedType:=GetType(PrimaryColors),
 ValueNameList:="red=Red|yellow=Yellow|blue=Blue"> _
```

Value	Name
“red”	“Red”
“yellow”	“Yellow”
“blue”	“Blue”

Example: Expanding an EnumeratedType

This uses the enumerated type shown in the **EnumeratedType** property. The user wants two more values, white and black.

[C#]

```
[DES.Enumerated(EnumeratedType=typeof(PrimaryColors),
 ValueNameList="white|black"]
```

[VB]

```
<DES.Enumerated(EnumeratedType:=GetType(PrimaryColors),
 ValueNameList:="white|black"> _
```

Value	Name
“red”	“red”
“yellow”	“yellow”
“blue”	“blue”
“white”	“white”
“black”	“black”

- **ListDelimiter** (char) – The delimiter between each name/value pair in **ValueNameList**.

  It defaults to the pipe character ("|").

- **ValueNameSeparator** (char) – The separator between each value and its name in the **ValueNameList** property.

  It defaults to the equals character ("=").

- **LookupIDLeadText** (string) – Uses the String Lookup System to define names. Each value already added to the list will be reviewed against the string group Misc to see if there is text to use there. As a result, you must setup an initial list using the **EnumeratedType** or **ValueNameList** properties.

  This property’s value is the first characters of the LookupID passed to the String Lookup System. It is followed by the value. For example, when using the enumerated type `HorizontalAlign`, **LookupIDLeadText**="`HorizAlign.`" and the enumerated element is "`Left`", the LookupID becomes “`HorizAlign.Left`”.

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

Example

Suppose the Misc string group has this text:

Key	Value
HorizAlign.NotSet	!!!
HorizAlign.Left	To the left
HorizAlign.Center	Centered
HorizAlign.Right	To the right

Note: The value !!! will delete an entry for NotSet.

[C#]

```
[DES.Enumerated(EnumeratedType=typeof(HorizontalAlign),
 LookupIDLeadText="HorizAlign."]
```

[VB]

```
<DES.Enumerated(EnumeratedType:=GetType(HorizontalAlign), _
 LookupIDLeadText:="HorizAlign."> _
```

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

Using the CustomizeColumn Method

You can programmatically define or edit the list of values on an EnumeratedStringAttribute by implementing the `PeterBlum.DES.DynamicData.ICustomizeColumn` interface on your EntityTable or the `PeterBlum.DES.DynamicData.IUICustomizeColumn` interface on your web form. See "Customizing Metadata At Runtime". Both define the `CustomizeColumn()` method.

The `CustomizeColumn()` method is passed a `PeterBlum.DES.DynamicData.ModifiableColumn` object which contains methods to modify an existing EnumeratedStringAttribute. Make sure the column already has the EnumeratedAttribute assigned. When `CustomizeColumn()` is called, the EnumeratedAttribute will already be populated with a list based on what was defined in the metadata.

Method: ModifyEnumeratedList

[C#]

```
public void ModifyEnumeratedList(string pValue, string pName, bool pApplyStringLookup)
```

[VB]

```
Public Sub ModifyEnumeratedList(ByVal pValue As String, _
      ByVal pName As String, ByVal pApplyStringLookup As String)
```

Parameters

pValue

The string value stored in the column. If the value already exists in the EnumeratedStringAttribute, it will be overwritten. Otherwise its added to the end of the list.

pName

The name shown to the user. If you pass "`!!!`", it will delete an entry with *pValue* that was previously defined. If you pass `null`/`Nothing`, it will use the same as *pValue*.

pApplyStringLookup

When `true`, use the String Lookup System to modify the value in *pName*. If the **LookupIDLeadText** property is defined on the EnumeratedAttribute and a key can be found in the String Lookup System, the value returned from the String Lookup System is used (except if blank). Otherwise the original value of *pName* is used.

Example

Populate the Region field of the Customer record with country names.

[C#]

```
[MetadataType(typeof(CustomerMetadata))]
public partial class Customer : DES.ICustomizeColumn
{
   public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn, bool pIsNewRow)
   {
      switch (pModifiableColumn.ColumnName)
      {
         case "Region":
            pModifiableColumn.ModifyEnumeratedList("USA", "United States", false);
            pModifiableColumn.ModifyEnumeratedList("Canada", null, false);
            pModifiableColumn.ModifyEnumeratedList("Mexico", null, false);
            break;
      }  // switch
   }
}
public class CustomerMetadata
{
   [DES.EnumeratedString()]
   public object Region { get; set; }
```

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

```
}
```

[VB]

```
<MetadataType(GetType(CustomerMetadata))> _
Partial Public Class Customer _
     Implements DES.ICustomizeColumn

   Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn,_
     ByVal pIsNewRow As Boolean)

      Select Case pModifiableColumn.ColumnName
         Case "Region"
            pModifiableColumn.ModifyEnumeratedList("USA", "United States", False)
            pModifiableColumn.ModifyEnumeratedList("Canada", Nothing, False)
            pModifiableColumn.ModifyEnumeratedList("Mexico", Nothing, False)
      End Select
   End Sub
End Class
Public Class CustomerMetadata
  <DES.EnumeratedString()> _
  Public Property Region As Object
     Get
       Return Nothing
     End Get
     Set
     End Set
  End Property
End Class
```

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

DES.UrlDataTypeAttribute

Class	PeterBlum.DES.DynamicData.UrlDataTypeAttribute
Supported Types	String representing a Url
Default Field Templates	Url.ascx, Url_Edit.ascx, and Url_Insert.ascx
Validation	RegExValidator confirming the format of a Url and optionally the file extensions.

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn". Be aware that you cannot create a DataTypeAttribute programmatically to establish a specific Field Template, but you can use it to effect a Validator on the Field Template.

- **AlwaysGenerateRegexValidator** (boolean) – Normally this attribute creates a RegexValidator to confirm the string is the correct format. It does not generate that validator when the same field has its own DES.RegularExpressionAttribute unless **AlwaysGenerateRegexValidator** is `true`.

  Assign it to `true` when you add a DES.RegularExpressionAttribute that has nothing to do with validating the overall format, such as it looks for specific elements within the Url.

  It defaults to `false`, allowing a DES.RegularExpressionAttribute to replace this attributes RegexValidator.

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the DataTypeCheckValidator. Assigns these properties on that validator. You can also setup error message properties using the Customizer control's **ErrorMessages** property.

- **Enabled** (boolean) – When `true`, the attribute is used. It defaults to `true`.

- **DisplayOrder** (int) – When multiple ValidationAttributes are used on a property, use this to determine the order they are shown both inline and in the ValidationSummary. The higher the value, the lower it appears. When not specified, it uses 0. If multiple ValidationAttributes have the same **DisplayOrder**, they are shown in the opposite order they are listed in the metadata.

- **UriScheme** (enum PeterBlum.DES.DynamicData.UrlDataTypeUriScheme) – Determines which of these Uri Schemes are allowed: http, https or ftp. Part of the validation rules which can be overridden by the **NameOfExpression** property.

  If you want to prevent the scheme from ever being entered, set **UriScheme** to None.

  The enumerated type `PeterBlum.DES.DynamicData.UrlDataTypeUriScheme` has these values.

  - `Any` - http, https, or ftp
  - `Web` - http or https. This is the default.
  - `http`
  - `https`
  - `ftp`
  - `None` - Entry of a Uri scheme is not allowed.

- **RequireUriScheme** (boolean) – Determines if the UriScheme element ("http://", "https://", "ftp://") is required in the URL. Part of the validation rules which can be overridden by the **NameOfExpression** property.

  If you want to prevent the scheme from ever being entered, set **UriScheme** to `None`.

  It defaults to `false`.

- **AllowPort** (boolean) – Determines if the port number can be entered in the URL. (Example: "www.domainname.com:80") Part of the validation rules which can be overridden by the **NameOfExpression** property.

  It defaults to `false`.

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

- **UriPartsAllowed** (enum PeterBlum.DES.DynamicData.UrlDataTypeUriPartsAllowed) – Determines if the URL can include file path and querystring parameters after the domain name. Part of the validation rules which can be overridden by the **NameOfExpression** property.

  The enumerated type PeterBlum.DES.DynamicData.UrlDataTypeUriPartsAllowed has these values:

  - `DomainOnly` - Nothing after the domain name and its extension.
  - `WithFile` - Allows folders and file name, but not querystring. This is the default.
  - `WithQueryString` - Allows folders, files, and querystring.

- **AllowIPAddress** (boolean) – Determines if the URL can include an IP address instead of a domain name. (Example: "http://135.20.135.20/folder/file.jpg") Part of the validation rules which can be overridden by the **NameOfExpression** property.

  It defaults to `false`.

- **NameOfExpression** (string) – This attribute creates a regular expression for the RegexValidator. If you want to use a different expression, create a new entry in `<RegExPatterns>` section of the **custom.des.config** file. Specify the name of the expression found in that file in this property.

- **FormattingExample** (string) – Shown in the validator's error message when it has the "{EXAMPLE}" token. It gives the user a formatting example. It will also be shown as hint on the Field Template's textbox when **ShowExampleAsHint** is `true` in the Field Template.

  Supports the token {DEFAULT} to show a predefined example.

  Supports the token {NEWLINE} for linebreaks.

  Supports HTML tags.

  If you want lead text like "Example: ", include it. For example, "Example: www.domain.com".

  It defaults to "{DEFAULT}".

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

- **SupportedFileExtensions** (string) – A second validator can be setup to limit the user to specific file extensions. This determines the list of file extentions and the **FileExtensionsErrorMessage** property determines the error message. When unassigned, no validation is applied to file extensions.

  Enter a pipe delimited list of supported file extensions (without the lead period). For example:

  `"jpg|gif|tif"`

  File extensions are case insensitive.

  When unassigned, no validation is applied to file extensions.

  It defaults to "".

- **FileExtensionsErrorMessage** (string) – When using the **SupportedFileExtensions** property, a second validator is created. This is the error message for that validator.

  Supports the token {EXTENSIONS} to show the list of extensions available.

  It defaults to `"The file must use one of these file extensions: {EXTENSIONS}"`.

- **FileExtensionsErrorMessageLookupID** (string) – Gets the value for **FileExtensionsErrorMessage** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should

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

be defined within the String Group of ErrorMessage. If no match is found OR this is blank, **FileExtensionsErrorMessage** 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.

- **FileExtensionsSummaryErrorMessage** (string) – When using the **SupportedFileExtensions** property, a second validator is created. This is the error message for that validator that is shown in the ValidationSummary control. **FileExtensionsErrorMessage** is used if this is unassigned.

  Supports the token {EXTENSIONS} to show the list of extensions available.

  It defaults to "".

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

Example

[C#]

```
[DES.UrlDataType(UriScheme=PeterBlum.DES.DynamicData.UrlDataTypeUriScheme.Any,
  RequireUriScheme=true,
  UriPartsAllowed=PeterBlum.DES.DynamicData.UrlDataTypeUriPartsAllowed.WithFile,
  SupportedFileExtensions="jpg|tif" )]
public object WebSite { get; set; }
```

[VB]

```
<DES.UrlDataType(UriScheme:=PeterBlum.DES.DynamicData.UrlDataTypeUriScheme.Any,_
  RequireUriScheme:=True, _
  UriPartsAllowed:=PeterBlum.DES.DynamicData.UrlDataTypeUriPartsAllowed.WithFile, _
  SupportedFileExtensions:="jpg|tif" )> _
Public Property WebSite As Object
  Get
    Return Nothing
  End Get
  Set
  End Set
End Property
```

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

DES.EmailAddressDataTypeAttribute

Class	PeterBlum.DES.DynamicData.EmailAddressDataTypeAttribute
Supported Types	String representing one or more email addresses
Default Field Templates	EmailAddress.ascx, EmailAddress_Edit.ascx, and EmailAddress_Insert.ascx
Validation	RegExValidator confirming the format of one or more email addresses. It will only allow multiple addresses when **MultipleAddressesAllowed** is `true`.

You can also add MX Record level validation. See "Providing additional validation of email addresses".

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn". Be aware that you cannot create a DataTypeAttribute programmatically to establish a specific Field Template, but you can use it to effect a Validator on the Field Template.

- **AlwaysGenerateRegexValidator** (boolean) – Normally this attribute creates a RegexValidator to confirm the string is the correct format. It does not generate that validator when the same field has its own DES.RegularExpressionAttribute unless **AlwaysGenerateRegexValidator** is `true`.

  Assign it to `true` when you add a DES.RegularExpressionAttribute that has nothing to do with validating the overall format, such as it looks for specific elements within the Url.

  It defaults to `false`, allowing a DES.RegularExpressionAttribute to replace this attributes RegexValidator.

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the DataTypeCheckValidator. Assigns these properties on that validator. You can also setup error message properties using the Customizer control's **ErrorMessages** property.

- **Enabled** (boolean) – When `true`, the attribute is used. It defaults to `true`.

- **DisplayOrder** (int) – When multiple ValidationAttributes are used on a property, use this to determine the order they are shown both inline and in the ValidationSummary. The higher the value, the lower it appears. When not specified, it uses 0. If multiple ValidationAttributes have the same **DisplayOrder**, they are shown in the opposite order they are listed in the metadata.

- **MultipleAddressesAllowed** (boolean) – Determines if the user can enter multiple email addresses, separated by the delimiter character defined in **DelimiterCharacter** and optionally followed by a space, when **DelimiterAllowsSpaces** is true. Part of the validation rules which can be overridden by the **NameOfExpression** property.

  It defaults to `false`.

- **DelimiterCharacter** (string) – When using **MultipleAddressesAllowed**, this is the delimiter between the addresses. It allows a single character, including a space. Part of the validation rules which can be overridden by the **NameOfExpression** property.

  It must always have a single character. If you permit spaces after the delimiter, set **DelimiterAllowsSpaces** to `true`.

  It defaults to a semicolon (`;`).

- **DelimiterAllowsSpaces** (boolean) – When using **MultipleAddressesAllowed**, if the user can type one or more spaces after the delimiter, set this to `true`. Part of the validation rules which can be overridden by the **NameOfExpression** property.

  It defaults to `false`.

- **NameOfExpression** (string) – This attribute creates a regular expression for the RegexValidator. If you want to use a different expression, create a new entry in `<RegExPatterns>` section of the **custom.des.config** file. Specify the name of the expression found in that file in this property.

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

- **FormattingExample** (string) – Shown in the validator's error message when it has the "{EXAMPLE}" token. It gives the user a formatting example. It will also be shown as hint on the Field Template's textbox when **ShowExampleAsHint** is `true` in the Field Template.

  Supports the token {DEFAULT} to show a predefined example.

  Supports the token {NEWLINE} for linebreaks.

  Supports HTML tags.

  If you want lead text like "Example: ", include it. For example, "Example: www.domain.com".

  It defaults to "{DEFAULT}".

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

- **SupportedFileExtensions** (string) – A second validator can be setup to limit the user to specific file extensions. This determines the list of file extentions and the **FileExtensionsErrorMessage** property determines the error message. When unassigned, no validation is applied to file extensions.

  Enter a pipe delimited list of supported file extensions (without the lead period). For example:

  `"jpg|gif|tif"`

  File extensions are case insensitive.

  When unassigned, no validation is applied to file extensions.

  It defaults to "".

- **FileExtensionsErrorMessage** (string) – When using the **SupportedFileExtensions** property, a second validator is created. This is the error message for that validator.

  Supports the token {EXTENSIONS} to show the list of extensions available.

  It defaults to `"The file must use one of these file extensions: {EXTENSIONS}"`.

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

- **FileExtensionsSummaryErrorMessage** (string) – When using the **SupportedFileExtensions** property, a second validator is created. This is the error message for that validator that is shown in the ValidationSummary control. **FileExtensionsErrorMessage** is used if this is unassigned.

  Supports the token {EXTENSIONS} to show the list of extensions available.

  It defaults to `""`.

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

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

Example

[C#]

```
[DES.UrlDataType(UriScheme=PeterBlum.DES.DynamicData.UrlDataTypeUriScheme.Any,
   RequireUriScheme=true,
   UriPartsAllowed=PeterBlum.DES.DynamicData.UrlDataTypeUriPartsAllowed.WithFile,
   SupportedFileExtensions="jpg|tif" )]
public object WebSite { get; set; }
```

[VB]

```
<DES.UrlDataType(UriScheme:=PeterBlum.DES.DynamicData.UrlDataTypeUriScheme.Any,_
   RequireUriScheme:=True, _
   UriPartsAllowed:=PeterBlum.DES.DynamicData.UrlDataTypeUriPartsAllowed.WithFile, _
   SupportedFileExtensions:="jpg|tif" )> _
Public Property WebSite As Object
   Get
      Return Nothing
   End Get
   Set
   End Set
End Property
```

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

Providing additional validation of email addresses

There are third party tools which can determine if the email address is geniune by using "MX records". Advanced Intellect's aspNetMX is one such tool.

Set up the ExtendedEmailValidation event handler when you are using a third party email address validation system. The method must be static (Shared in VB).

You must assign the `PeterBlum.DES.DynamicData.IExtendedEmailAddressValidation` interface on the entity table class (not the entity table metadata class). Use the method to analyze the email address text and return `true` if it's valid.

Your method will only be called if the text was not blank and the built-in pattern evaluation approved the text.

Here is the definition of the delegate:

[C#]

```
public delegate bool ExtendedEmailValidation(object pSender,
 string pEmailText);
```

[VB]

```
Public Delegate Function ExtendedEmailValidation(_
 ByVal pSender As Object, ByVal pEmailText As String) As Boolean
```

Parameters

pSender

The EmailAddressCondition object that is getting evaluated.

pEmailText

The text from the TextBox. Evaluate this string to confirm that it is a valid email address.

Return value

When `true`, the pEmailText is a valid email address. When `false`, it is not.

Example

This example uses a fictional object SuperEmailTest to determine if pEmailText is valid or not. Its `Validate()` method returns `true` when it is valid. If you are using Advanced Intellect's aspNetMX product, see below for explicit instructions.

[C#]

```
public class Customers : DES.IExtendedEmailAddressValidation
{
   public static bool ExtendedEmailValidation(object pSender, string pEmailText)
   {
      return SuperEmailTest.Validate(pEmailText);  // fictional engine
   }
}
```

[VB]

```
Public Class Customers Inherits DES.IExtendedEmailAddressValidation
   Public Shared Function ExtendedEmailValidation ( _
      ByVal pSender As Object, ByVal pEmailText As String) As Boolean _
         Implements DES.IExtendedEmailAddressValidation.ExtendedEmailValidation
      Return SuperEmailTest.Validate(pEmailText)   ' fictional engine
   End Function

End Class
```

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

DES.UrlAsImageDataTypeAttribute

Class	PeterBlum.DES.DynamicData.UrlAsImageDataTypeAttribute
Supported Types	String representing a URL
Default Field Templates	UrlAsImage.ascx, UrlAsImage_Edit.ascx, and UrlAsImage_Insert.ascx
Validation	RegExValidator confirming the file extension is a type of image.

Using this Attribute

A URL can be shown as a string. However, when it points to an image, the browser can display that image. If you want only to display a string, use DES.DataType(DataType.Url). If you want to display an image, use the UrlAsImageDataTypeAttribute.

The UrlAsImage Field Template is flexible, showing the image in a reduced size and optionally letting the user click it to popup an expanded view of the image.

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn". Be aware that you cannot create a DataTypeAttribute programmatically to establish a specific Field Template, but you can use it to effect a Validator on the Field Template.

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the DataTypeCheckValidator. Assigns these properties on that validator. You can also setup error message properties using the Customizer control's **ErrorMessages** property.
- **SupportedFileTypes** (string) – A pipe delimited list of supported file extensions (without the lead period). It defaults to: `"jpg|jpe|jpeg|bmp|gif|tif|png"`.

  If you want to add other extensions, follow the file extension with "=" + [MIME type]. This will determine the MIME type for the new extension. (All existing extensions have known MIME types so they don't require the additional element.)

  For example: `"jpg|jpe|jpeg|ico=image/x-icon"`.
- **ExpandedViewUserControlUrl** (string) – The URL to a UserControl that displays the image in an "expanded view" (usually full size). When assigned, the user can click on the image to display the Expanded View.

  When "", a click on the image does nothing.

  It defaults to "".

  Here are ExpandedView usercontrols shipped with DES Dynamic Data:

  `"~/DynamicData/Ajax Control Toolkit/ExpandedViewForImages.ascx"` - Provides a modal dialog designed for images using the Ajax Control Toolkit's ModalPopupExtender. Only available if you have installed support for the Ajax Control Toolkit. (To install, run the Web Application Updater with the "Update a web application" option. When it gets to the Third Party Controls panel, make sure you select the Ajax Control Toolkit.)

  If you develop an ExpandedView UserControl, it must implement the `PeterBlum.DES.DynamicData.Futures.IExpandedViewUI` interface. [!!!]

  The UrlAsImage Field Template also has a **ExpandedViewUserControlUrl** property, so you can set it up in the user interface layer with a DynamicField or DynamicControl control.
- **UrlFormat** (string) – The text to assign to the Image's **ImageUrl** property on the Field Template. Use the token "{FIELD}" to insert the text from the column. If there is no token, it appends the text from the column. Use this when your stored Url is missing some elements of the path, like the domain name. The format will contain those missing elements.

  It defaults to "".

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

- **DisplayWidth** (integer) – The suggested width of the image, in pixels, as shown on the page. When 0, the actual image width is used. The actual Field Template has its own property to set the width, called **ImageWidth**. **DisplayWidth** is only used when **ImageWidth** is 0.
- **DisplayHeight** (integer) – The suggested height of the image, in pixels, as shown on the page. When 0, the actual image height is used. The actual Field Template has its own property to set the height, called **ImageHeight**. **DisplayHeight** is only used when **ImageHeight** is 0.
- **Enabled** (boolean) – When `true`, the attribute is used. It defaults to `true`.
- **DisplayOrder** (int) – When multiple ValidationAttributes are used on a property, use this to determine the order they are shown both inline and in the ValidationSummary. The higher the value, the lower it appears. When not specified, it uses 0. If multiple ValidationAttributes have the same **DisplayOrder**, they are shown in the opposite order they are listed in the metadata.

Example

[C#]

```
[DES.UrlAsImageDataType(ExpandedViewUserControlUrl=
   "~/DynamicData/Ajax Control Toolkit/ExpandedViewForImages.ascx")]
public object PhotoPath { get; set; }
```

[VB]

```
<DES.UrlAsImageDataType(ExpandedViewUserControlUrl:= _
   "~/DynamicData/Ajax Control Toolkit/ExpandedViewForImages.ascx")> _
Public Property PhotoPath As Object
   Get
      Return Nothing
   End Get
   Set
   End Set
End Property
```

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

DES.DbImageDataTypeAttribute

Class	PeterBlum.DES.DynamicData.DbImageDataTypeAttribute
Supported Types	Binary containing an image file
Default Field Templates	DbImage.ascx, DbImage_Edit.ascx, and DbImage_Insert.ascx
Validation	RegExValidator confirming the file extension is a type of image. Note: You can provide virus detection validation with the DetectVirus event handler.

Using this Attribute

When storing images in your database, you generally use a Binary type column. In addition, it helps to have a second column that tracks uniqueness, by using a date/time stamp of the last time it was saved. The date/time stamp helps signal the browser that a URL and image file it has cached is no longer valid as that stamp is inserted into the Url sent to the browser.

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn". Be aware that you cannot create a DataTypeAttribute programmatically to establish a specific Field Template, but you can use it to effect a Validator on the Field Template.

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the DataTypeCheckValidator. Assigns these properties on that validator. You can also setup error message properties using the Customizer control's **ErrorMessages** property.
- **VirusErrorMessage** (string) – ErrorMessage text used when Virus detection code detects a virus. Virus detection requires adding your own code that uses the API of your antivirus software through the VirusDetection.DetectVirus event handler. See "Adding Virus Detection to uploaded files".

  Supports the token "{FILENAME}" to show the filename retrieved.

  It defaults to "`Virus detected in the file [{FILENAME}]. Upload aborted.`".
- **VirusErrorMessageLookupID** (string) – Gets the value for **VirusErrorMessage** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of ErrorMessage. If no match is found OR this is blank, **VirusErrorMessage** 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.
- **BadFormatErrorMessage** (string) – ErrorMessage text used when the file is not a valid image.

  Supports the token "{FILENAME}" to show the filename retrieved.

  It defaults to "`The file [{FILENAME}] is not the correct format. Upload aborted.`".
- **BadFormatErrorMessageLookupID** (string) – Gets the value for **BadFormatErrorMessage** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of ErrorMessage. If no match is found OR this is blank, **BadFormatErrorMessage** 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.
- **RequiredErrorMessage** (string) – ErrorMessage text assigned when the column requires a value but does not have one. If not assigned, it falls back to the RequiredAttribute value.

  It defaults to "`The field {LABEL} must be assigned a file.`".

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

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

- **SupportedFileTypes** (string) – A pipe delimited list of supported file extensions (without the lead period). It defaults to: `"jpg|jpe|jpeg|bmp|gif|tif|png"`.

  If you want to add other extensions, follow the file extension with "=" + [MIME type]. This will determine the MIME type for the new extension. (All existing extensions have known MIME types so they don't require the additional element.)

  For example: `"jpg|jpe|jpeg|ico=image/x-icon"`.

- **DisplayWidth** (integer) – The suggested width of the image, in pixels, as shown on the page. When 0, the actual image width is used. The actual Field Template has its own property to set the width, called **ImageWidth**. **DisplayWidth** is only used when **ImageWidth** is 0.

- **DisplayHeight** (integer) – The suggested height of the image, in pixels, as shown on the page. When 0, the actual image height is used. The actual Field Template has its own property to set the height, called **ImageHeight**. **DisplayHeight** is only used when **ImageHeight** is 0.

- **DisplayInEdit** (Boolean) – When true, show the image in edit mode. When false, do not show the image. Just offer the FileUpload control. The actual Field Template has its own property called **DisplayInEdit**. This property is only used when the Field Template's **DisplayInEdit** is `null`/`Nothing`.

- **ExpandedViewUserControlUrl** (string) – The URL to a UserControl that displays the image in an "expanded view" (usually full size). When assigned, the user can click on the image to display the Expanded View.

  When "", a click on the image does nothing.

  It defaults to "".

  Here are ExpandedView usercontrols shipped with DES Dynamic Data:

  `"~/DynamicData/Ajax Control Toolkit/ExpandedViewForImages.ascx"` - Provides a modal dialog designed for images using the Ajax Control Toolkit's ModalPopupExtender. Only available if you have installed support for the Ajax Control Toolkit. (To install, run the Web Application Updater with the "Update a web application" option. When it gets to the Third Party Controls panel, make sure you select the Ajax Control Toolkit.)

  If you develop an ExpandedView UserControl, it must implement the `PeterBlum.DES.DynamicData.Futures.IExpandedViewUI` interface. See "Creating Expanded View UserControls".

  The UrlAsImage Field Template also has a **ExpandedViewUserControlUrl** property, so you can set it up in the user interface layer with a DynamicField or DynamicControl control.

- **TimeStampColumnName** (string) – Optional name specifying a column containing a DateTime value or long integer that reflects a unique identity of the value stored in the Image's column. This allows the user to change the image and have browser's caches see that change by including an additional element in the QueryString that comes from this column.

  Field Templates based in PeterBlum.DES.DynamicData.DbImageEditFTUC will automatically support this. They update the TimeStamp column with either the current date time or DateTime converted into a long integer, depending on the column's type.

  If not supplied, no time stamp is used to monitor changes.

  It defaults to "".

- **Enabled** (boolean) – When `true`, the attribute is used. It defaults to `true`.

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

Example

[C#]

```
[DES.DbImageDataType(ExpandedViewUserControlUrl=
   "~/DynamicData/Ajax Control Toolkit/ExpandedViewForImages.ascx")]
public object Picture { get; set; }
```

[VB]

```
<DES.DbImageDataType(ExpandedViewUserControlUrl:= _
   "~/DynamicData/Ajax Control Toolkit/ExpandedViewForImages.ascx")> _
Public Property Picture As Object
   Get
      Return Nothing
   End Get
   Set
   End Set
End Property
```

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

ValidationAttributes: Applying Constraints on Columns

Validation is the most common business rule on a database. Each column needs to have constraints on the data it accepts and validation supplies those constraints. For example, an integer column for Age may be constrained to values from 0 to 100.

In traditional ASP.NET web form development, you build these validation rules by using the Validator web controls, found in ASP.NET and **Peter's Professional Validation**. That approach detaches the validation on the web form from the rules intended by the database developer. It's easy for the web form developer to create the wrong rules for a column. Dynamic Data places the validation rules on the column by assigning ValidationAttributes to properties on your EntityTable Metadata classes.

The class `System.ComponentModel.DataAnnotations.ValidationAttribute` is the base class for all ValidationAttributes. Its subclasses define properties specific to the validation rule. For example, the RangeAttribute validates a range with properties for **Minimum** and **Maximum**.

Click on any of these topics to jump to them:

- Standard types vs DES types
- ErrorMessages on ValidationAttributes
  - ErrorMessage Properties found on DES ValidationAttributes
- How Field Templates utilize the ValidationAttributes
- The types of ValidationAttributes supplied with DES Dynamic Data
- Using the CustomizeColumn Method
- Writing server side only Validation Code within your EntityTable Classes
- Invoking Validation prior to saving

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

Standard types vs DES types

ASP.NET Dynamic Data defines four ValidationAttribute subclasses: RequiredAttribute, RangeAttribute, RegularExpressionAttribute, and StringLengthAttribute. DES Dynamic Data also provides its own versions of these attributes, as it expands their features. DES's versions are preferred although not usually required. DES uses the same class name as the native attributes, but it uses a different namespace.

When assigning ValidationAttributes to columns (properties) on the EntityTable Metadata Classes, here's how to differentiate between the two.

Native attribute	[C#]
	`[ValidationAttributeName()]`
	[VB]
	`<ValidationAttributeName()> _`

DES attribute	[C#]
	`[DES.ValidationAttributeName()]`
	[VB]
	`<DES.ValidationAttributeName()> _`

Native namespace	`System.ComponentModel.DataAnnotations.ValidationAttributeName`

DES namespace	`PeterBlum.DES.DynamicData.ValidationAttributeName`
	Your Metadata classes file should declare:
	[C#]
	`uses DES=PeterBlum.DES.DynamicData`
	[VB]
	`Imports DES=PeterBlum.DES.DynamicData`

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

ErrorMessages on ValidationAttributes

Field Templates use ValidationAttributes to generate Validator web controls. ValidationAttributes optionally supply the **ErrorMessage** property to those web controls. When using DES's ValidationAttributes, they can also supply **SummaryErrorMessage**, **ErrorMessageLookupID**, and **SummaryErrorMessageLookupID** properties found on DES's Validators. Yet, these are not necessarily the right places to define these values because ValidationAttributes are declared within your business logic, and error messages are a user interface concept.

DES Dynamic Data lets you set up error messages in several places to address a separation of concerns.

1. Web Form developers use the Customizer control with ts **ErrorMessages** property. Define a list of validator types and their associated error messages. When used, it overrides any of the other sources for error messages.
2. ValidationAttribute supplies the **ErrorMessage** properties already mentioned.
3. If you do not setup either the Customizer or the **ErrorMessage** properties on the ValidationAttribute, the ValidationAttribute subclass will supply a default. *This default is in English. If you need to localize, you must use one of the previous options.*

Any Field Templates that support validation (edit or insert mode) should have a ColumnValidatorManager control. It defines the ErrorFormatter (a user interface to display error messages) and can handle when there are multiple errors to show at once. The **DisplayOrder** property on DES's ValidationAttributes can assist it order the error messages shown.

ErrorMessage Properties found on DES ValidationAttributes

Each of these properties can be defined as parameters on any of DES's ValidationAttributes.

For example:

[C#]

```
[DES.RangeValidator(Minimum="10", Maximum="20",
  ErrorMessage="Enter a value between {MINIMUM} and {MAXIMUM}."
  SummaryErrorMessageLookupID="SEM.ColumnNameRange" ]
```

[VB]

```
<DES.RangeValidator(Minimum:="10", Maximum:="20", _
  ErrorMessage:="Enter a value between {MINIMUM} and {MAXIMUM}." _
  SummaryErrorMessageLookupID:="SEM.ColumnNameRange" > _
```

- **ErrorMessage** (string) – Text displayed at the validator's location on the page through its ErrorFormatter. This value is copied to the Validator's **ErrorMessage** property.

  Supports the same tokens found on the validator created by this ValidationAttribute. For example, when using the CompareToValueAttribute, it creates the CompareToValueValidator which offers the token "{VALUETOCOMPARE}".

  The token "{LABEL}" is always available to show the column name. (The DisplayNameAttribute can be used to create a user friendly name.)

  Note: ASP.NET Dynamic Data supports tokens in the ValidationAttribute. They differ from those used by DES, such as {0} for the label. DES supports most of those tokens but its recommended you use DES's tokens for clarity.

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

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

- **SummaryErrorMessage** (string) – Supplies the error message for the ValidationSummary control and the alert shown when using **PeterBlum.DES.Globals.Page.ShowAlertOnSubmit**. You can leave it blank if the value of **ErrorMessage** is the message you want in the ValidationSummary and alert.

  SummaryErrorMessage supports the same tokens and editor in the Properties Editor as **ErrorMessage** does.

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

- **DisplayOrder** (integer) – When multiple ValidationAttributes are used on a property, use this to determine the order they are shown both inline and in the ValidationSummary. The higher the value, the lower it appears. When not specified, it uses 0. If multiple ValidationAttributes have the same **DisplayOrder**, they are shown in the opposite order they are listed in the metadata.

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

How Field Templates utilize the ValidationAttributes

Field Templates provide the user interface for a single data type. They must convert the ValidationAttributes into DES Validator controls. DES's ColumnValidatorManager control does the work. Add it to the page and pass its reference to the `SetUpColumnValidatorManager()` method.

This differs from ASP.NET Dynamic Data, which has you declare all possible validators directly on the page (and leaves the possibility a validator will not be defined that is declared in the metadata.) Yet, you can manually create validators if desired. Pass their references to the `SetUpDESValidator()` method.

Step by Step instructions

1. Setup these capabilities on Field Templates for Edit and Insert modes. If the Field Template does not edit data, you can skip these steps.

2. The UserControl should declare this line at the top:

```
<%@ Register tagprefix="desDD" namespace="PeterBlum.DES.DynamicData"
assembly="PeterBlum.DES.DynamicData" %>
```

3. Add the ColumnValidatorManager control to the Field Template. Its position will be where the error messages are displayed. So generally it positioned after the editable controls. The ColumnValidatorManager does all of the conversion from ValidationAttributes to Validator controls.

```
<desDD:ColumnValidatorManager ID="ColumnValidatorManager1" runat="server" />
```

4. In the `Page_Init()` method, call the method `SetUpColumnValidatorManager()`, passing the ColumnValidatorManager control.

   [C#]

```
protected void Page_Init(object sender, EventArgs e)
{
   SetUpEditableDataControl(editablecontrol);
   SetUpColumnValidatorManager(ColumnValidatorManager1);
}
```

   [VB]

```
Protected Sub Page_Init(ByVal sender As Object, ByVal e As EventArgs)
   SetUpEditableDataControl(editablecontrol)
   SetUpColumnValidatorManager(ColumnValidatorManager1)
End Sub
```

5. You can define the ErrorFormatter (user interface for displaying error messages) directly on the ColumnValidatorManager control's **ErrorFormatter** property. See the section "ErrorFormatters: Customizing the appearance of the Error Message" in the **Validation User's Guide**. If you do not, it will use a default supplied by the PageManager control's **DefaultErrorFormatterSkinID** property.

   #### Example: Set the default to a PopupErrorFormatter

   The Master Page contains this code. This example also sets the **ErrorFormatter.Display** property to `Dynamic`.

```
<des:PageManager ID="DESPageManager" runat="server"
   AJAXFramework="MicrosoftAJAX"
   DefaultErrorFormatterSkinID="PopupView {Display='Dynamic'}" />
```

   For details on the **DefaultErrorFormatterSkinID** property, see the topic "ErrorFormatter Skins" in the section "ErrorFormatters: Customizing the appearance of the Error Message" of the **Validation User's Guide**.

6. *Unusual case:* If you manually define a DES Validator control in the Field Template, call the method `SetUpDESValidator()`, passing the Validator control in `Page_Init()` after the call to `SetUpColumnValidatorManager()`.

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

The types of ValidationAttributes supplied with DES Dynamic Data

Click on the Type to jump to its description and properties.

ValidationAttribute Type	DES Validator it creates
DES.RequiredAttribute	RequiredTextValidator or RequiredSelectionValidator depending on the data entry control used by the Field Template *Note: When LINQ to SQL identifies that the column is required in the ColumnAttribute, it will automatically create the Required validator.*
DES.RangeAttribute	RangeValidator
DES.CompareToValueAttribute	CompareToValueValidator
DES.RegularExpressionAttribute	RegExValidator
DES.StringLengthAttribute	TextLengthValidator *Note: When LINQ to SQL defines a length in the ColumnAttribute, it will automatically create the TextLengthValidator.*
DES.CompareTwoColumnsAttribute	CompareTwoFieldsValidator
DES.DifferenceAttribute	DifferenceValidator
DES.StringListAttribute	CompareToStringsValidator
DES.InjectionSecurityAttribute	FieldSecurityValidator
DES.CharacterSetAttribute	CharacterValidator *Note: Field Templates also apply this to their FilteredTextBox controls.*
DES.DateDataTypeAttribute, **DES.TimeOfDayDataTypeAttribute,** **DES.DurationDataTypeAttribute,** **DES.AnniversaryDataTypeAttribute,** **DES.MonthYearDataTypeAttribute,** **DES.IntegerDataTypeAttribute,** **DES.DecimalDataTypeAttribute,** **DES.CurrencyDataTypeAttribute,** **DES.PercentDataTypeAttribute,** **DES.IntegerMeasurementDataTypeAttribute,** **DES.DecimalMeasurementDataTypeAttribute**	DataTypeCheckValidator with its **DataType** property set appropriately. *Note: DataTypeAttributes also selects a Field Template. See "DataTypeAttributes: The DataTypes used by your Business logic".*
DES.UrlDataTypeAttribute **DES.UrlAsImageDataTypeAttribute** **DES.DbImageDataTypeAttribute**	RegexValidator
DES.EmailAddressDataTypeAttribute	EmailAddressValidator
DES.CreditCardNumberAttribute	CreditCardNumberValidator
DES.ABARoutingNumberAttribute	ABARoutingNumberValidator

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

Using the CustomizeColumn Method

You can programmatically define or edit ValidationAttributes by implementing the `PeterBlum.DES.DynamicData.ICustomizeColumn` interface on your EntityTable or `PeterBlum.DES.DynamicData.IUICustomizeColumn` interface on your web form. Both define the `CustomizeColumn()` method. See "Customizing Metadata At Runtime". `CustomizeColumn()` gives you full access to the data of the current row being delivered to the page. So you can base your validation rules on live data.

The `CustomizeColumn()` method is passed a `PeterBlum.DES.DynamicData.ModifiableColumn` object which contains methods to retrieve the desired ValidationAttribute. Choose from these methods on ModifiableColumn to add or edit a ValidationAttribute object.

- `GetExistingValidationAttribute()` – Returns the ValidationAttribute of the type you supply. If that attribute is not defined on the column's metadata, it returns `null`/`Nothing`. See "Method: GetExistingValidationAttribute".
- `GetValidationAttribute()` – Returns the ValidationAttribute of the type you supply. If that attribute is not defined on the column's metadata, it creates the ValidationAttribute with default property values and returns that object for you to modify. See "Method: GetValidationAttribute".
- `RemoveValidationAttribute()` – Removes the ValidationAttribute of the type you supply. See "Method: RemoveValidationAttribute".

Method: GetExistingValidationAttribute

Use when the desired ValidationAttribute has been defined in the column's metadata and you want to modify it or even disable it. You can disable any DES ValidationAttribute (but not native ValidationAttribute) by setting its **Enabled** property to `false`.

Note: It does NOT support the native ValidationAttributes from System.ComponentModel.DataAnnotations.

Note: Edits made to the ValidationAttribute here only affect the Field Template for the current row of data being generated.

[C#]

```
public System.ComponentModel.DataAnnotations.ValidationAttribute
         GetExistingValidationAttribute(Type pValidationAttributeType)
```

[VB]

```
Public Function GetExistingValidationAttribute(ByVal pValidationAttributeType As Type)_
         As System.ComponentModel.DataAnnotations.ValidationAttribute
```

Parameters

pValidationAttributeType

The type of ValidationAttribute you are seeking. It only supports types that implement `PeterBlum.DES.DynamicData.IDESValidationAttribute`, which include all DES's ValidationAttributes.

Returns

Returns the ValidationAttribute of the type you supply. If that attribute is not defined on the column's metadata, it returns `null`/`Nothing`. You should typecast the result to the specific class requested by *pValidationAttributeType*.

EXAMPLE ON THE NEXT PAGE

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

Example: Modifying a Validator's property

On the Employee table, the user has assigned the CompareToValueAttribute to the HireDate column. This date value must not exceed today's date.

[C#]

```
[MetadataType(typeof(EmployeeMetadata))]
public partial class Employee : DES.ICustomizeColumn
{
   public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn, bool pIsNewRow)
   {
      switch (pModifiableColumn.ColumnName)
      {
         case "HireDate":
            DES.CompareToValueAttribute vCTV =
               pModifiableColumn.GetExistingValidationAttribute(
                  typeof(DES.CompareToValueAttribute));
            vCTV.ValueToCompare = DateTime.Today;
            break;
      }  // switch
   }
}
public class EmployeeMetadata
{
   [DES.CompareToValue()]      // ValueToCompare assigned in CustomizeColumn method
   public object HireDate { get; set; }
}
```

[VB]

```
<MetadataType(GetType(EmployeeMetadata))> _
Partial Public Class Employee _
     Implements DES.ICustomizeColumn

   Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn,_
     ByVal pIsNewRow As Boolean)

      Select Case pModifiableColumn.ColumnName
         Case "HireDate"
            Dim vCTV As DES.CompareToValueAttribute = _
               pModifiableColumn.GetExistingValidationAttribute( _
                  GetType(DES.CompareToValueAttribute))
            vCTV.ValueToCompare = DateTime.Today
      End Select
   End Sub
End Class
Public Class EmployeeMetadata
   <DES.CompareToValue()> _ ' ValueToCompare assigned in CustomizeColumn method
   Public Property HireDate As Object
     Get
       Return Nothing
     End Get
     Set
     End Set
   End Property
End Class
```

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

Example: Disable a ValidationAttribute

On the Employee table, the user has assigned the RegularExpressionAttribute to HomePhone. It reflects a pattern that supports United States and Canada phone numbers. For other countries, disable the RegularExpressionAttribute.

Note: This is not a real world example because the validator should be created with an Enabler that disables it as the user edits data. That case needs a DependencyAttribute.

[C#]

```
[MetadataType(typeof(EmployeeMetadata))]
public partial class Employee : DES.ICustomizeColumn
{
   public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn, bool pIsNewRow)
   {
      switch (pModifiableColumn.ColumnName)
      {
         case "HomePhone":
     // look at the Country column for its current value
            if ((this.Country != "United States") && (this.Country != "Canada"))
            {
               DES.RegularExpressionAttribute vREV =
                  pModifiableColumn.GetExistingValidationAttribute(
                     typeof(DES.RegularExpressionAttribute));
               vREV.Enabled = false;
            }
            break;
      }  // switch
   }
}
public class EmployeeMetadata
{
  [DES.RegularExpression(NameOfExpression="U.S. Phone")]
  public object HomePhone { get; set; }
}
```

[VB]

```
<MetadataType(GetType(EmployeeMetadata))> _
Partial Public Class Employee _
     Implements DES.ICustomizeColumn

   Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn,_
     ByVal pIsNewRow As Boolean)

      Select Case pModifiableColumn.ColumnName
         Case "HomePhone"
     ' look at the Country column for its current value
            If (Me.Country <> "United States") And (Me.Country <> "Canada") Then
               Dim vREV As DES.RegularExpressionAttribute = _
                  pModifiableColumn.GetExistingValidationAttribute( _
                     GetType(DES.RegularExpressionAttribute))
               vREV.Enabled = False
            End If
      End Select
   End Sub
End Class
```

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

```
Public Class EmployeeMetadata
  <DES.RegularExpression(NameOfExpression:="U.S. Phone") >_
  Public Property HireDate As Object
    Get
      Return Nothing
    End Get
    Set
    End Set
  End Property
End Class
```

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

Method: GetValidationAttribute

Use when you want to create the ValidationAttribute or use the existing ValidationAttribute when its already defined in the column's metadata. This overloaded method has two forms.

Note: It does NOT support the native ValidationAttributes from System.ComponentModel.DataAnnotations.

Note: Edits made to the ValidationAttribute here only affect the Field Template for the current row of data being generated.

[C#]

```
public System.ComponentModel.DataAnnotations.ValidationAttribute
         GetValidationAttribute(Type pValidationAttributeType)

public System.ComponentModel.DataAnnotations.ValidationAttribute
         GetValidationAttribute(Type pValidationAttributeType,
          out bool pWasCreated)
```

[VB]

```
Public Function GetExistingValidationAttribute(ByVal pValidationAttributeType As Type)_
         As System.ComponentModel.DataAnnotations.ValidationAttribute

Public Function GetExistingValidationAttribute(ByVal pValidationAttributeType As Type, _
          ByRef pWasCreated As Boolean)_
         As System.ComponentModel.DataAnnotations.ValidationAttribute
```

Parameters

pValidationAttributeType

The type of ValidationAttribute you are seeking. It only supports types that implement `PeterBlum.DES.DynamicData.IDESValidationAttribute`, which include all DES's ValidationAttributes.

pWasCreated

When you need to know if the ValidationAttribute was created (not previously defined), use this parameter. It returns `true` when it was created and `false` when it was previously defined.

Returns

Returns the ValidationAttribute of the type you supply. If that attribute is not defined on the column's metadata, it returns `null`/`Nothing`. You should typecast the result to the specific class requested by *pValidationAttributeType*.

EXAMPLE ON THE NEXT PAGE

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

Example: Creating a ValidatorAttribute property

On the Employee table, add the CompareToValueAttribute to the HireDate column. This date value must not exceed today's date. The attribute is not already defined in the EmployeeMetadata class.

[C#]

```
[MetadataType(typeof(EmployeeMetadata))]
public partial class Employee : DES.ICustomizeColumn
{
   public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn, bool pIsNewRow)
   {
      switch (pModifiableColumn.ColumnName)
      {
         case "HireDate":
            DES.CompareToValueAttribute vCTV =
               pModifiableColumn.GetValidationAttribute(
                  typeof(DES.CompareToValueAttribute));
            vCTV.ValueToCompare = DateTime.Today;
            break;
      }  // switch
   }
}
public class EmployeeMetadata
{
  public object HireDate { get; set; }
}
```

[VB]

```
<MetadataType(GetType(EmployeeMetadata))> _
Partial Public Class Employee _
     Implements DES.ICustomizeColumn

   Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn,_
     ByVal pIsNewRow As Boolean)

      Select Case pModifiableColumn.ColumnName
         Case "HireDate"
            Dim vCTV As DES.CompareToValueAttribute = _
               pModifiableColumn.GetValidationAttribute( _
                  GetType(DES.CompareToValueAttribute))
            vCTV.ValueToCompare = DateTime.Today
      End Select
   End Sub
End Class
Public Class EmployeeMetadata
  Public Property HireDate As Object
     Get
       Return Nothing
     End Get
     Set
     End Set
  End Property
End Class
```

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

Method: RemoveValidationAttribute

Remove the ValidationAttribute set by metadata. (It does not affect the global metadata for this column.)

[C#]

```
public void RemoveValidationAttribute()
```

[VB]

```
Public Sub RemoveValidationAttribute()
```

Example

Remove the existing RequiredAttribute on the PostalCode field of the Customer table.

[C#]

```
[MetadataType(typeof(CustomerMetadata))]
public partial class Customer : DES.ICustomizeColumn
{
   public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn, bool pIsNewRow)
   {
      switch (pModifiableColumn.ColumnName)
      {
         case "PostalCode":
            pModifiableColumn.RemoveValidationAttribute(
               typeof(DES.RequiredAttribute));
            break;
      }  // switch
   }
}
public class CustomerMetadata
{
   [DES.Required()]
   public object PostalCode { get; set; }
}
```

[VB]

```
<MetadataType(GetType(CustomerMetadata))> _
Partial Public Class Customer _
     Implements DES.ICustomizeColumn

   Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn,_
     ByVal pIsNewRow As Boolean)

      Select Case pModifiableColumn.ColumnName
         Case "PostalCode"
            pModifiableColumn.RemoveValidationAttribute( _
               GetType(DES.RequiredAttribute))
      End Select
   End Sub
End Class
Public Class CustomerMetadata
   <DES.Required()> _
   Public Property PostalCode As Object
      Get
         Return Nothing
      End Get
      Set
      End Set
   End Property
End Class
```

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

Writing server side only Validation Code within your EntityTable Classes

While ValidationAttributes cover many validation cases, your business logic may be more complex, requiring some specialzied code. For example, a new record cannot be added if the Name column has the same value as an entry in your table. ASP.NET Dynamic Data allows you to create validation code within the EntityTable classes. If your code detects an error, it should throw a `System.ComponentModel.DataAnnotations.ValidationException` with an error message. The error is routed to either the FieldTemplate's ColumnValidatorManager control or the ValidationSummary control depending on if it's field specific or not.

ASP.NET 4 introduced the CustomValidationAttribute and the Validator class. These work together to both simplify the process and give you more flexible ways to communicate errors. ASP.NET 4 users can still use the methodology of calling the ValidaitonException explicitly, as described in "Using ASP.NET 3.5".

Click on any of these topics to jump to them:

- Using ASP.NET 4: The CustomValidationAttribute
  - Understanding the CustomValidationAttribute declaration
  - Understanding the method definition
- Using ASP.NET 3.5

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

Using ASP.NET 4: The CustomValidationAttribute

The `System.ComponentModel.DataAnnotations.CustomValidationAttribute` is used for any server-side only validation that you could not implement through any other ValidationAttribute. This attribute supports validation on individual properties and on the overall Entity class object (checking several properties together).

The validator's logic is placed in a method in your Entity class (or another class of your choice). It's invoked by calling the `System.ComponentModel.DataAnnotations.Validator.TryValidateObject()` or `TryValidateProperty()` methods. Generally you will call `TryValidateObject()` from the Entity's method that does a final validation. See "Invoking Validation prior to saving".

Example

This is an example using the Category table from Northwind DB and LINQ to SQL.

[C#]

```
[MetadataType(typeof(CategoryMetadata))]
[CustomValidation(typeof(Category), "FinalCheck")]
public partial class Category
{
   public class CategoryMetadata
   {
      [CustomValidation(typeof(Category), "TestCategoryName")]
      public object CategoryName { get; set; }
   }
   // method defined by LINQ to SQL on every Entity class.
   partial void OnValidate(System.Data.Linq.ChangeAction action)
   {
      List<ValidationResult> vErrors = new List<ValidationResult>();

      // validate class-level attributes
      // When the last parameter is true, it also evaluates all ValidationAttributes
      // on this entity's properties
      Validator.TryValidateObject(this, new ValidationContext(this, null, null), vErrors, true);

      // handle errors
      if (vErrors.Count > 0)
         throw new ValidationException(vErrors[0], null, this);
   }

   public static ValidationResult FinalCheck(Category pCategory, ValidationContext pValidationContext)
   {
      if (pCategory.Description.Length < 10)
         return new ValidationResult("Description must be at least 10 characters.",
               new List<string> { "Description" });
       return ValidationResult.Success;
   }
   public static ValidationResult TestCategoryName(string pNewName,
       ValidationContext pValidationContext)
   {
      if (Regex.IsMatch(pNewName, @"^\d")) // cannot start with a digit
         return new ValidationResult("Cannot start with a digit", new List<string> { "CategoryName" });
      return ValidationResult.Success;
   }
}
```

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

[VB]

```
<MetadataType(GetType(CategoryMetadata))> _
<CustomValidation(GetType(Category), "FinalCheck")> _
Public Partial Class Category
   Public Class CategoryMetadata
      Private _CategoryName As Object
      <CustomValidation(GetType(Category), "TestCategoryName")> _
      Public Property CategoryName() As Object
         Get
            Return _CategoryName
         End Get
         Set
            _CategoryName = value
         End Set
      End Property
   End Class
   ' method defined by LINQ to SQL on every Entity class.
   Private Partial Sub OnValidate(ByVal action As System.Data.Linq.ChangeAction)
      Dim vErrors As New List(Of ValidationResult)()

     ' validate class-level attributes
     ' When the last parameter is true, it also evaluates all ValidationAttributes
     ' on this entity's properties
      Validator.TryValidateObject(Me, New ValidationContext(Me, Nothing, Nothing), vErrors, True)

      ' handle errors
      If vErrors.Count > 0 Then
         Throw New ValidationException(vErrors(0), Nothing, Me)
      End If
   End Sub

   Public Shared Function FinalCheck(ByVal pCategory As Category, _
      ByVal pValidationContext As ValidationContext) As ValidationResult
      If pCategory.Description.Length < 10 Then
         Return New ValidationResult("Description must be at least 10 characters.",
            New List(Of String)("Description"))
      End If
      Return ValidationResult.Success
   End Function

   Public Shared Function TestCategoryName(ByVal pNewName As String, _
      ByVal pValidationContext As ValidationContext) As ValidationResult
      If Regex.IsMatch(pNewName, "^\d") Then ' cannot start with a digit
         Return New ValidationResult("Cannot start with a digit", New List(Of String)( "CategoryName"))
      End If
      Return ValidationResult.Success
   End Function
End Class
```

Understanding the CustomValidationAttribute declaration

[C#]

```
[CustomValidation(typeof(class that owns the method), "method name")]
```

[VB]

```
<CustomValidation(GetType(class that owns the method), "method name")> _
```

- It can be placed on the class or property definition. Both are shown above.
- The class that contains your validation method must always be declared in the CustomValidationAttribute declaration, even if the method is in the same class this attribute is assigned.
- You may want to create a separate class for all validation rules of your entity, such as “class CategoryValidation”.
- The method name property must be a case sensitive match your method name.

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

Understanding the method definition

There are two available method signatures for the method invoked by the CustomValidationAttribute.

[C#]

```
using System.ComponentModel.DataAnnotations;
…
public static ValidationResult MethodName(object value)
public static ValidationResult MethodName(object value, ValidationContext context)
```

[VB]

```
Imports System.ComponentModel.DataAnnotations
…
Public Shared Function MethodName(value As Object) As ValidationResult
Public Shared Function MethodName(value As Object, context As ValidationContext)
  As ValidationResult
```

Use the one parameter method when validating a property.

Use the two parameter method when validating at the class level or when you need access to the Entity instance because it has other properties that apply to your validation rules.

The method itself must be static/shared and public.

Parameters

Value

When the method is associated with the class, it will be passed an instance of the Entity. It can be typed to match the class of the Entity, such as it is with "Category" above. It can also be typed to "object".

```
public static ValidationResult FinalCheck(Category pCategory,
   ValidationContext pValidationContext)
```

```
Public Shared Function FinalCheck(ByVal pCategory As Category, _
   ByVal pValidationContext As ValidationContext) As ValidationResult
```

When the method is associated with a property, it will be passed the value of the property. It can be typed to match the type of the property, such as it is with "string" above. It can also be typed to "object".

```
public static ValidationResult TestCategoryName(string pNewName,
   ValidationContext pValidationContext)
```

```
Public Shared Function TestCategoryName(ByVal pNewName As String, _
   ByVal pValidationContext As ValidationContext) As ValidationResult
```

ValidationContext

The `System.ComponentModel.DataAnnotations.ValidationContext` class describes other details about what's being validated. It's more important when validating a property since it supplies the instance of the Entity (**ObjectInstance** property) and the property name (**MemberName** property).

When the CustomValidationAttribute is on a property and it needs access to another property, typecast the ObjectInstance property to your Entity's class to access the other property.

[C#]

```
if (pHireDate > ((Employee) pValidationContext.ObjectInstance).TerminationDate)
```

[VB]

```
If pHireDate > (CType(pValidationContext.ObjectInstance, _
 Employee).TerminationDate Then
```

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

Returns

Always return an instance of System.ComponentModel.DataAnnotations.ValidationResult. If there is an error, set the ValidationResult.ErrorMessage property with a description of the error. Optionally return a list of parameter names.

[C#]

```
return new ValidationResult("Cannot start with a digit",
  new List<string> { "CategoryName" });
```

[VB]

```
Return New ValidationResult("Cannot start with a digit", _
  New List(Of String)( "CategoryName"))
```

When there is no error, just return the static property **ValidationResult.Success**, as shown above. Otherwise return a description and optionally the property name(s) with errors.

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

Using ASP.NET 3.5

Open the code file generated by Linq To SQL or the ADO.NET Entity Framework. Each table class has a series of partial methods that start with "On". In particular, notice those that end with the same name as your columns.

- When using LINQ to SQL, add code into the `OnValidate()` method to perform validation that is not specific to any one property. Throw a ValidationException if an error is detected.
- When using ADO.NET EntityFramework, set up the `SavingChanges` event handler to perform validation that is not specific to any one property. Throw a ValidationException if an error is detected.

See "How to: Customize Data Field Validation" for details.

Example using LINQ to SQL

Note: If you are using ADO.NET Entity Framework, the ideas are the same. You will be using the SavingChanges event handler instead of OnValidate().

```
public partial class Category : INotifyPropertyChanging, INotifyPropertyChanged
{
  partial void OnValidate(System.Data.Linq.ChangeAction action);

  partial void OnCategoryNameChanging(string value);
  partial void OnDescriptionChanging(string value);
}
```

Note: VB Examples are shown at the end of this section.

You will be cloning these methods and adding your validation code within your cloned methods. Your code will go into the other code file containing the other partial EntityTable class that also contains your metadata. (Since there are three "EntityTable" classes in for each table, lets be specific. You will be adding to the class that has the MetadataTypeAttribute above it.)

Your task is to evaluate the parameter and throw an exception if its not a legal value.

For example:

```
[MetadataType(typeof(CategoryMetadata))]
public partial class Category
{
  partial void OnCategoryNameChanging(string value)
  {
  // use a regular expression to invalid several useless terms
    if (System.Text.RegularExpressions.Regex.IsMatch(value,
      "^((temp)|(category)|(asdg))$",
      System.Text.RegularExpressions.RegexOptions.IgnoreCase))
        throw new ValidationException("[" + value + "] is not allowed.");
  }
}
```

When you need to evaluate several property values at once, such as "if CategoryName has XYZ and CategoryDescription has ABC", use the `OnValidate()` partial method that is also defined for you in the EntityTable class, as shown above.

Again, clone its method and add it to the other partial EntityTable class. Write your code, evaluating the properties of the class and throwing a ValidationException when there is an error. This exception will be relayed to DynamicValidator on the Page Template or created by the DES DynamicDataManager control.

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

```
[MetadataType(typeof(CategoryMetadata))]
public partial class Category
{
  partial void OnValidate(System.Data.Linq.ChangeAction action)
  {
    if ((this.CategoryName == "XYZ") && (this.Description == "ABC"))
      throw new ValidationException("Do not enter both XYZ and ABC");
  }
}
```

The above code in VB

```
Partial Public Class Category
  Inherits INotifyPropertyChanging
  Implements INotifyPropertyChanged

  Partial Private Sub OnValidate(ByVal action As System.Data.Linq.ChangeAction)
  End Sub

  Partial Private Sub OnCategoryNameChanging(ByVal value As String)
  End Sub
  Partial Private Sub OnDescriptionChanging(ByVal value As String)
  End Sub
End Class
```

```
<MetadataType(GetType(CategoryMetadata))>  _
Partial Public Class Category
  Partial Private Sub OnCategoryNameChanging(ByVal value As String)
  ' use a regular expression to invalid several useless terms
    If System.Text.RegularExpressions.Regex.IsMatch(value, _
      "^((temp)|(category)|(asdg))$", _
      System.Text.RegularExpressions.RegexOptions.IgnoreCase) Then
      Throw New ValidationException("[" + value + "] is not allowed.")
    End If
  End Sub

  Partial Private Sub OnValidate(ByVal action As System.Data.Linq.ChangeAction)
    If (Me.CategoryName = "XYZ")  _
        AndAlso (Me.Description = "ABC") Then
      Throw New ValidationException("Do not enter both XYZ and ABC")
    End If
  End Sub
End Class
```

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

Invoking Validation prior to saving

Your EntityTable classes should implement a validation method that is called prior to being saved. This method run validation code build into every ValidationAttribute (including DataTypeAttributes and CustomValidationAttribute). This ensures that if the consumer of your EntityTable (such as the user interface or web service) does not fully handle validation, you still have a way to validate.

Use `System.ComponentModel.DataAnnotations.Validator.TryValidateObject()` to do the job. Always pass `true` as the last parameter to insure the properties are validated. (Warning: If you pass `false` or omit the parameter, it will only validate the RequiredAttributes on properties.)

Your method's implementation depends upon whether you are using LINQ to SQL or Entity Framework, and whether you are using Data Access Objects to handle your CRUD operations. Here is the code used by any Entity validation method:

[C#]

```
List<ValidationResult> vErrors = new List<ValidationResult>();

ValidationContext vValidationContext = new ValidationContext(this, null, null);
Validator.TryValidateObject(this, vValidationContext, vErrors, true);

if (vErrors.Count > 0)
   throw new ValidationException(vErrors[0], null, this);
```

[VB]

```
Dim vErrors As New List(Of ValidationResult)()

Dim vValidationContext As ValidationContext = _
   New ValidationContext(Me, Nothing, Nothing)
Validator.TryValidateObject(Me, vValidationContext, vErrors, True)
If vErrors.Count > 0 Then
   Throw New ValidationException(vErrors(0), Nothing, Me)
End If
```

Documentation for classes shown above

- ValidationResult
- ValidationContext
- ValidationException

Working Data Access Objects from Versatile DataSources

The EntityDAO classes within Versatile DataSources fully implement the Validator.TryValidateObject() method and call it from their Update and Insert methods. This is supported whether you use LINQ to SQL, Entity Framework, or ADO.NET.

You do not need to do anything else.

Working with the Domain object from Domain Services

Implement the call to Validator.TryValidateObject() in each Update and Insert method coded into your DomainObject class. Call it first.

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

Working with LINQ to SQL and no Data Access Object

LINQ to SQL declares the OnValidate() method where you should call Validator.TryValidateObject(). LINQ to SQL automatically invokes this method. You do not need to write code to call it.

[C#]

```
[MetadataType(typeof(CategoryMetadata))]
public partial class Category
{
  partial void OnValidate(System.Data.Linq.ChangeAction action)
  {
    List<ValidationResult> vErrors = new List<ValidationResult>();

    ValidationContext vValidationContext = new ValidationContext(this, null, null);
    Validator.TryValidateObject(this, vValidationContext, vErrors, true);

    if (vErrors.Count > 0)
      throw new ValidationException(vErrors[0], null, this);
  }
}
```

[VB]

```
Partial Private Sub OnValidate(ByVal action As System.Data.Linq.ChangeAction)
  Dim vErrors As New List(Of ValidationResult)()

  Dim vValidationContext As ValidationContext = _
    New ValidationContext(Me, Nothing, Nothing)
  Validator.TryValidateObject(Me, vValidationContext, vErrors, True)

  If vErrors.Count > 0 Then
    Throw New ValidationException(vErrors(0), Nothing, Me)
  End If
End Sub
```

Working with Entity Framework and no Data Access Object

When using ADO.NET EntityFramework, set up the **SavingChanges** event handler.

See “How to: Execute Business Logic When Saving Changes”.

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

DES.RequiredAttribute

Class	PeterBlum.DES.DynamicData.RequiredAttribute
Validation	RequiredTextValidator or RequiredSelectionValidator, depending on the type of data entry control on the Field Template.
Default ErrorMessage	`"The field {LABEL} is required."`

Using this Attribute

Require a value in the column.

If your database already marks the column as required ("cannot be null"), LINQ to SQL and ADO.NET Entity Framework will automatically have ASP.NET Dynamic Data create this attribute for you. If you also define it on a column marked required, your RequiredAttribute will override the attribute that was automatically created. Use this technique to supply any of the properties shown below.

If this validation rule can be enabled and disabled based on some condition, such as when another column has a value, also use a DependencyAttribute on this column's metadata. See "DependencyAttributes: Relationships between columns".

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn Method".

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the Validator control. See "ErrorMessages on ValidationAttributes".
- **UnassignedValues** (string) – The textual value of the control that means it is unassigned. When the control's value matches this value, the condition indicates failure. Matching may be case sensitive or insensitive based on **UnassignedCaseInsensitive**. It defaults to "".
- **UnassignedCaseInsenstive** (Boolean) - Determines if the **UnassignedValues** property is matched case insensitive or not. When `true`, it is case insensitive. It defaults to `true`.
- **Enabled** (boolean) – When `true`, the ValidationAttribute is used. It defaults to `true`. Generally you set this to `false` using the `CustomizeColumn()` method when the ValidationAttribute was defined in metadata but will not be used on the current row.
- **DisplayOrder** (int) – When multiple ValidationAttributes are used on a property, use this to determine the order they are shown both inline and in the ValidationSummary. The higher the value, the lower it appears. When not specified, it uses 0. If multiple ValidationAttributes have the same **DisplayOrder**, they are shown in the opposite order they are listed in the metadata.

Example

[C#]

```
[DES.Required()]
```

[VB]

```
<DES.Required()> _
```

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

DES.RangeAttribute

Class	PeterBlum.DES.DynamicData.RangeAttribute
Validation	RangeValidator.
Default ErrorMessage	`"The field {LABEL} must be between {MINIMUM} and {MAXIMUM}."`

Using this Attribute

Establish a range. Generally ranges are applied to integer, decimal, date, time and duration values, although any type supported by the RangeValidator's DataType property is allowed.

Set the range using the **Minimum** and **Maximum** properties.

If this validation rule can be enabled and disabled based on some condition, such as when another column has a value, also use a DependencyAttribute on this column's metadata. See "DependencyAttributes: Relationships between columns".

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn Method".

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the Validator control. See "ErrorMessages on ValidationAttributes".
- **Minimum** (object) – The minimum value of the range. As a object type, it accepts any type.

  When assigning the parameters of the attribute in the column's metadata, assign integer and decimal values for numbers. For dates, use a string in the format "yyyy-DD-mm". For times and durations, use the format "HH:mm:ss". *Note: .net does not allow runtime values like properties from the DateTime object in attributes.*
- **Maximum** (object) - The maximum value of the range. As a object type, it accepts any type.

  When assigning the parameters of the attribute in the column's metadata, assign integer and decimal values for numbers. For dates, use a string in the format "yyyy-DD-mm". For times and durations, use the format "HH:mm:ss". *Note: .net does not allow runtime values like properties from the DateTime object in attributes.*
- **Enabled** (boolean) – When `true`, the ValidationAttribute is used. It defaults to `true`. Generally you set this to `false` using the `CustomizeColumn()` method when the ValidationAttribute was defined in metadata but will not be used on the current row.
- **DisplayOrder** (int) – When multiple ValidationAttributes are used on a property, use this to determine the order they are shown both inline and in the ValidationSummary. The higher the value, the lower it appears. When not specified, it uses 0. If multiple ValidationAttributes have the same **DisplayOrder**, they are shown in the opposite order they are listed in the metadata.

Example: Range on a Date type column

[C#]

```
[DES.Range(Minimum="1900-01-01", Maximum="1990-12-31")]
```

[VB]

```
<DES.Range(Minimum:="1900-01-01", Maximum:="1990-12-31") > _
```

Example: Range on an Integer type column

[C#]

```
[DES.Range(Minimum=1, Maximum=20)]
```

[VB]

```
<DES.Range(Minimum:=1, Maximum:=20) > _
```

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

DES.CompareToValueAttribute

Class	PeterBlum.DES.DynamicData.CompareToValueAttribute
Validation	CompareToValueValidator
Default ErrorMessage	`"The field {LABEL} must be " + Operator + " {VALUETOCOMPARE}."`

Using this Attribute

Compare the value of this column to a value assigned to the **ValueToCompare** property.

If this validation rule can be enabled and disabled based on some condition, such as when another column has a value, also use a DependencyAttribute on this column's metadata. See "DependencyAttributes: Relationships between columns".

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn".

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the Validator control. See "ErrorMessages on ValidationAttributes".
- **ValueToCompare** (object) – The value to compare. As a object type, it accepts any type.

  When assigning the parameters of the attribute in the column's metadata, assign integer and decimal values for numbers. For dates, use a string in the format "yyyy-DD-mm". For times and durations, use the format "HH:mm:ss". *Note: .net does not allow runtime values like properties from the DateTime object in attributes.*
- **Operator** (enum PeterBlum.DES.ConditionOperator) – Determines how the data entry control is compared to **ValueToCompare**. The enumerated type `PeterBlum.DES.ConditionOperator` has the following values:
  - `Equal` (This is the default.) column value = ValueToCompare
  - `NotEqual` column value <> ValueToCompare
  - `GreaterThan` column value > ValueToCompare
  - `GreaterThanEqual` column value >= ValueToCompare
  - `LessThan` column value < ValueToCompare
  - `LessThanEqual` column value <= ValueToCompare
- **Enabled** (boolean) – When `true`, the ValidationAttribute is used. It defaults to `true`. Generally you set this to `false` using the `CustomizeColumn()` method when the ValidationAttribute was defined in metadata but will not be used on the current row.
- **DisplayOrder** (int) – When multiple ValidationAttributes are used on a property, use this to determine the order they are shown both inline and in the ValidationSummary. The higher the value, the lower it appears. When not specified, it uses 0. If multiple ValidationAttributes have the same **DisplayOrder**, they are shown in the opposite order they are listed in the metadata.

Example: Integer must be less than 10

[C#]

```
[DES.CompareToValue(ValueToCompare=10,
  pOperator=PeterBlum.DES.ConditionOperator.LessThan)]
```

[VB]

```
<DES.CompareToValue(ValueToCompare:=10, _
  pOperator:=PeterBlum.DES.ConditionOperator.LessThan)> _
```

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

DES.RegularExpressionAttribute

Class	PeterBlum.DES.DynamicData.RegularExpressionAttribute
Validation	RegexValidator

Default ErrorMessage	`"The field {LABEL} is not valid."`
	or
	`"The field {LABEL} is not a valid " + NameOfExpression + "."`

Using this Attribute

Match a string column value to a specific pattern using a regular expression. You can either assign an expression to the **Expression** property or use the name of a predefined expression in the **NameOfExpression** property. Predefined expressions are found in the `<RegExPatterns>` section of the **des.config** and **custom.des.config** files.

If this validation rule can be enabled and disabled based on some condition, such as when another column has a value, also use a DependencyAttribute on this column's metadata. See "DependencyAttributes: Relationships between columns".

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn".

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the Validator control. See "ErrorMessages on ValidationAttributes".
- **NameOfExpression** (string) – Gets a regular expression declared in the **des.config** or **custom.des.config** file's `<RegExpPatterns>` section. If not found, an exception is thrown. If the **Expression** property is also assigned, it overrides NameOfExpression. It defaults to "".

  Here are the predefined expression names that come with DES:

  - U.S. Phone
  - International Phone
  - Any Phone
  - French Phone
  - Japanese Phone
  - German Phone
  - P.R.C Phone
  - Email Address
  - Web URL
  - MailTo URL
  - FTP URL
  - Any URL
  - U.S. Zip Code (5 and 9 digits)
  - US and Canadian Postal Codes
  - Canadian Postal Code
  - UK Postal Code, space separator required
  - UK Postal Code, space separator optional
  - French Postal Code
  - German Postal Code
  - Japanese Postal Code
  - P.R.C Postal Code
  - Credit Card with spaces
  - Credit Card without spaces
  - U.S. Social Security Number
  - P.R.C Social Security (ID)

- **Expression** (string) – The regular expression. Use one that is valid with both .Net and JavaScripts regular expression parsers. See JavaScript's definition of regular expressions.
- **CaseInsensitive** (Boolean) – When `true`, the regular expression parser will perform a case insensitive match. When `false`, it will perform a case sensitive match. It defaults to `true`.
- **Multiline** (Boolean) – When `true`, it changes the meaning of the regular expression symbols "^" and "$" so they match at the beginning and end, respectively, of any line, and not just the beginning and end of the entire string. It defaults to `false`.

  WARNING: Some browsers do not support the Multiline flag on client-side validation. This includes IE Mac 5.x and Netscape 4.x. These browsers will abort processing the client-side JavaScript condition code when this option is detected. Use this option with care.

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

- **IgnoreBlankText** (Boolean) – Determines how blank text is evaluated. When `true`, the condition cannot be evaluated. When `false`, the condition evaluates as failed (reports an error). It defaults to `true`.
- **SupportsJavaScriptParser** (Boolean) – The server side uses the .net framework regular expression parser while the client side uses JavaScript. JavaScript's syntax is more limited. If you develop an expression beyond the capabilities of javascript, set this to false to only use the .net framework parser.

  It defaults to `true`, allowing the expression to be used by Javascript.

  If you intend to develop expressions that work with JavaScript, use this reference: JavaScript RegExp class guide.
- **Enabled** (boolean) – When `true`, the ValidationAttribute is used. It defaults to `true`. Generally you set this to `false` using the `CustomizeColumn()` method when the ValidationAttribute was defined in metadata but will not be used on the current row.
- **DisplayOrder** (int) – When multiple ValidationAttributes are used on a property, use this to determine the order they are shown both inline and in the ValidationSummary. The higher the value, the lower it appears. When not specified, it uses 0. If multiple ValidationAttributes have the same **DisplayOrder**, they are shown in the opposite order they are listed in the metadata.

Example

[C#]

```
[DES.RegularExpression(NameOfExpression="Web URL")]
```

[VB]

```
<DES.RegularExpression(NameOfExpression:="Web URL") > _
```

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

DES.StringLengthAttribute

Class	PeterBlum.DES.DynamicData.StringLengthAttribute
Validation	TextLengthValidator
Default ErrorMessage	`"The field {LABEL} not exceed {MAXIMUM} characters."`

Using this Attribute

Impose limits on the number of characters permitted in a string column. Use **Maximum** to define a maximum number of characters and **Minimum** for a minimum limit.

If your database already defines the number of characters limit on the column, LINQ to SQL and ADO.NET Entity Framework will automatically have ASP.NET Dynamic Data create this attribute for you. If you also define it on a column marked required, your StringLengthAttribute will override the attribute that was automatically created. Use this technique to supply any of the properties shown below.

If this validation rule can be enabled and disabled based on some condition, such as when another column has a value, also use a DependencyAttribute on this column's metadata. See "DependencyAttributes: Relationships between columns".

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn".

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the Validator control. See "ErrorMessages on ValidationAttributes".
- **Minimum** (Integer) – The minimum number of characters. If 0, it is not used.
- **Maximum** (Integer) – The maximum number of characters. If 0, it is not used.
- **Enabled** (boolean) – When `true`, the ValidationAttribute is used. It defaults to `true`. Generally you set this to `false` using the `CustomizeColumn()` method when the ValidationAttribute was defined in metadata but will not be used on the current row.
- **DisplayOrder** (int) – When multiple ValidationAttributes are used on a property, use this to determine the order they are shown both inline and in the ValidationSummary. The higher the value, the lower it appears. When not specified, it uses 0. If multiple ValidationAttributes have the same **DisplayOrder**, they are shown in the opposite order they are listed in the metadata.

Example

[C#]

```
[DES.StringLength(Maximum=100)]
```

[VB]

```
<DES.StringLength(Maximum:=100)> _
```

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

DES.CompareTwoColumnsAttribute

Class	PeterBlum.DES.DynamicData.CompareTwoColumnsAttribute
Validation	CompareTwoFieldsValidator
Default ErrorMessage	`"The field {LABEL} must be " + Operator + " {LABEL2}."`

Using this Attribute

Compares the value of this column with another column. Always specify the name of the other column in the **SecondColumnName** property. Use the **Operator** property to establish the comparison.

If this validation rule can be enabled and disabled based on some condition, such as when another column has a value, also use a DependencyAttribute on this column's metadata. See "DependencyAttributes: Relationships between columns".

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn".

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the Validator control. See "ErrorMessages on ValidationAttributes".
- **SecondColumnName** (string) – The name of the column in this table that will be compared. Do not use the value from DisplayNameAttribute, if it exists. This property requires a valid column name or an exception will occur.
- **Operator** (enum PeterBlum.DES.ConditionOperator) – Determines how the value of this column is compared to the **SecondColumnName** column. The enumerated type `PeterBlum.DES.ConditionOperator` has the following values:
  - `Equal` (This is the default.) this column value = second column value
  - `NotEqual` this column value <> second column value
  - `GreaterThan` this column value > second column value
  - `GreaterThanEqual` this column value >= second column value
  - `LessThan` this column value < second column value
  - `LessThanEqual` this column value <= second column value
- **Enabled** (boolean) – When `true`, the ValidationAttribute is used. It defaults to `true`. Generally you set this to `false` using the `CustomizeColumn()` method when the ValidationAttribute was defined in metadata but will not be used on the current row.
- **DisplayOrder** (int) – When multiple ValidationAttributes are used on a property, use this to determine the order they are shown both inline and in the ValidationSummary. The higher the value, the lower it appears. When not specified, it uses 0. If multiple ValidationAttributes have the same **DisplayOrder**, they are shown in the opposite order they are listed in the metadata.

Example

[C#]

```
[DES.CompareToColumn(SecondColumnName="EndDate",
   Operator=PeterBlum.DES.ConditionOperator.LessThanEqual)]
```

[VB]

```
<DES.CompareToColumn(SecondColumnName:="EndDate", _
   Operator:=PeterBlum.DES.ConditionOperator.LessThanEqual) > _
```

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

DES.DifferenceAttribute

Class	PeterBlum.DES.DynamicData.DifferenceAttribute
Validation	DifferenceValidator
Default ErrorMessage	`" The field {LABEL} must be " + Operator + " {LABEL2} by {DIFFERENCEVALUE}. The difference is currently {DIFFRESULT}."`

Using this Attribute

Compare the value of two columns to determine how the difference between their values compares to a value in **DifferenceValue**. Always specify the name of the other column in the **SecondColumnName** property. Use the **Operator** property to establish the comparison.

If this validation rule can be enabled and disabled based on some condition, such as when another column has a value, also use a DependencyAttribute on this column's metadata. See "DependencyAttributes: Relationships between columns".

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn".

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the Validator control. See "ErrorMessages on ValidationAttributes".
- **SecondColumnName** (string) – The name of the column in this table that will be compared. Do not use the value from DisplayNameAttribute, if it exists. This property requires a valid column name or an exception will occur.
- **Operator** (enum PeterBlum.DES.ConditionOperator) – Determines how the calculated difference is compared to **DifferenceValue**. The enumerated type `PeterBlum.DES.ConditionOperator` has the following values:
  - `Equal` (This is the default.) abs(this column value – second column value) = DifferenceValue
  - `NotEqual` abs(this column value – second column value) <> DifferenceValue
  - `GreaterThan` abs(this column value – second column value) > DifferenceValue
  - `GreaterThanEqual` abs(this column value – second column value) >= DifferenceValue
  - `LessThan` abs(this column value – second column value) < DifferenceValue
  - `LessThanEqual` abs(this column value – second column value) <= DifferenceValue
- **DifferenceValue** (Double) – The value to be compared to the difference between the two values. (The difference is always an absolute value.) It must be 0 or higher. (When set to 0, you have effectively made a CompareTwoFieldsCondition.)

  Dates are compared as a number of days between the two columns. For example, the difference between January 1, 2003 and January 2, 2003 is one day. Times are compared as a number of seconds.

  This value is the right side of the expression. The calculated difference is on the left side.
- **Enabled** (boolean) – When `true`, the ValidationAttribute is used. It defaults to `true`. Generally you set this to `false` using the `CustomizeColumn()` method when the ValidationAttribute was defined in metadata but will not be used on the current row.
- **DisplayOrder** (int) – When multiple ValidationAttributes are used on a property, use this to determine the order they are shown both inline and in the ValidationSummary. The higher the value, the lower it appears. When not specified, it uses 0. If multiple ValidationAttributes have the same **DisplayOrder**, they are shown in the opposite order they are listed in the metadata.

Example

[C#]

```
[DES.Difference(SecondColumnName="EndDate", DifferenceValue=10,
  Operator=PeterBlum.DES.ConditionOperator.GreaterThanEqual)]
```

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

[VB]

```
<DES.CompareToColumn(SecondColumnName:="EndDate", DifferenceValue:=10,_
  Operator:=PeterBlum.DES.ConditionOperator.GreaterThanEqual) > _
```

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

DES.StringListAttribute

Class	PeterBlum.DES.DynamicData.StringListAttribute
Validation	CompareToStringsValidator
Default ErrorMessage	`" The field {LABEL} must contain one if these items: {1}."`

Using this Attribute

Use on string columns that have a specific list of string values. Assign a delimited list of strings to the **List** property. The delimiter defaults to the pipe (`|`) character but can be changed with the **Delimiter** property. The strings can be the entire value or a partial value of the column as determined by the **MatchTextRule** property.

If this validation rule can be enabled and disabled based on some condition, such as when another column has a value, also use a DependencyAttribute on this column's metadata. See "DependencyAttributes: Relationships between columns".

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn".

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the Validator control. See "ErrorMessages on ValidationAttributes".

  Use the token "{1}" to display the list of strings. It uses the value from `PeterBlum.DES.Globals.Page.CultureInfo.TextInfo.ListSeparator` to separate items in the list.

- **List** (string) – A delimited list of strings to match to the column value. The delimited defaults to the pipe (`|`) character but can be changed using the **Delimiter** property. When set, it replaces the current contents of the Items collection.

- **Delimiter** (char) – The delimiter character used to convert the string in the List property into a list of strings. It defaults to the pipe (`|`) character.

- **Items** (System.Collections.Generic.List<string>) – Accessible within the ICustomizeColumn.CustomizeColumn() method. The same data as in the List property, but in a collection. It will be updated when the List property is assigned

- **CaseInsenstive** (Boolean) - Determines if the string matching is case insensitive or not. When `true`, it is case insensitive. It defaults to `true`.

- **MatchTextRule** (enum PeterBlum.DES.MatchTextRule) – Determines how the value of the text field is compared to each string in the list. This enumerated type has these values:
  - `Exact` – The value must exactly match a string. This is the default.
  - `Contains` – The value must be somewhere within a string.
  - `StartsWith` – The value must appear at the start of a string.
  - `EndsWith` – The value must appear at the end of a string.

- **Enabled** (boolean) – When `true`, the ValidationAttribute is used. It defaults to `true`. Generally you set this to `false` using the `CustomizeColumn()` method when the ValidationAttribute was defined in metadata but will not be used on the current row.

- **DisplayOrder** (int) – When multiple ValidationAttributes are used on a property, use this to determine the order they are shown both inline and in the ValidationSummary. The higher the value, the lower it appears. When not specified, it uses 0. If multiple ValidationAttributes have the same **DisplayOrder**, they are shown in the opposite order they are listed in the metadata.

Example

[C#]

```
[DES.StringList(List="red|green|blue")]
```

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

[VB]

```
<DES.StringList(List:="red|green|blue")> _
```

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

DES.InjectionSecurityAttribute

Class	PeterBlum.DES.DynamicData.InjectionSecurityAttribute
Validation	FieldSecurityValidator
Default ErrorMessage	`"The field {LABEL} contains illegal text. Text must not contain programming code using " + ["SQL" or "HTML" or "Javascript"] + "."`

Using this Attribute

ALERT: Requires the Peter's Input Security module to be setup.

Look for signs of hackers employing SQL Injection and/or Cross Site Scripting attacks. See the **Input Security User's Guide** for details on hacking defenses. If your page uses the PageSecurityValidator and the column permits some HTML, add this to define what HTML is permitted. Otherwise, your users will get security errors.

If this validation rule can be enabled and disabled based on some condition, such as when another column has a value, also use a DependencyAttribute on this column's metadata. See "DependencyAttributes: Relationships between columns".

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn".

Please refer to the **Input Security User's Guide** for more details on the properties of the FieldSecurityValidator.

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the Validator control. See "ErrorMessages on ValidationAttributes".
- **DetectSQLInjection** (Boolean) – Use this to detect SQL injection attacks. It uses the SQL Detection Engine.

  When `true`, it detects SQL injection. It defaults to `true`.

  Set it to `false` only if you fully neutralize the input and do not want to report an error when SQL expressions are supplied.

  The Security Analysis Report may also recommend setting this to `false` because it finds other validators that will protect your page. When enabled, Peter's Input Security uses CPU intensive code to detect attacks. If you want to log SQL injection attacks, set this to `true` because the **Peter's Data Entry Suite** validators cannot log attacks.

  Use the **SQLDetectionLevel** property to determine how aggressive the testing is.

  To learn more about and customize the SQL Detection Engine, see "About the SQL Detection Engine" in the **Input Security User's Guide**.

  *Note: All possible characters are permitted by the SQL Detection Engine. If you want to prevent them, use a CharacterValidator from **Peter's Data Entry Suite**.*

- **SQLDetectionLevel** (enum PeterBlum.DES.Security.SQLDetectionLevel) – When detecting SQL injection, this determines how aggressive the testing is. In most cases, this should be set to `High`. If you set it to anything else, anticipate some SQL expressions to get through. Be sure your code includes the proper neutralization code.

  This enumerated type has these values:

  - `High` – Suggested use: any field that never permits any SQL keyword.

    This is the default setting for this property.
  - `MediumHigh` – Suggested use: fields that allow short phrases or names with the occasional SQL keyword. For example, city name, street address, and person's name. Single line text that is usually short enough not to allow useful SQL statements.
  - `Medium` – Suggested use: single line free-form fields that do not exceed 100 characters. (The idea is short entries without carriage returns.)

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

- `MediumLow` – Suggested use: multiline free-form fields that never include SQL statements.
- `Low` – Suggested use: multiline free-form fields that sometimes may have valid SQL statements.

All of these levels still look for a few common hacking patterns and will reject any term defined in the `<illegalsqlelements>` section of the Peter's Input Security configuration files. For more, see "What Each SQL Detection Level Uses To Detect Attacks" in the **Input Security User's Guide**.

If you need to allow SQL statements, disable detection of injection and neutralize the data.

- **DetectScriptInjection** (Boolean) – Use this to detect script injection attacks. It uses the HTML and Script Detection Engine.

  When `true`, detect script injection. It defaults to `true`.

  Set it to `false` only if you fully neutralize the input and do not want to report an error when HTML and script oriented expressions are supplied.

  The Security Analysis Report may also recommend setting this to `false` because it finds other validators that will protect your page. When enabled, Peter's Input Security uses CPU intensive code to detect attacks. If you want to log script injection attacks, set this to `true` because the **Peter's Data Entry Suite** validators cannot log attacks.

  Use the **HTMLTagMode** and **HTMLTags** properties to determine what tags are considered valid.

  To learn more about and customize the SQL Detection Engine, see "About The HTML and Script Detection Engine" in the **Input Security User's Guide**.

  Note: Most characters are permitted by the HTML and Script Detection Engine. If you want to prevent them, use a CharacterValidator from DES's Validation Framework.

- **HTMLTagMode** (enum PeterBlum.DES.Security.HTMLTagMode) – When detecting script injection attacks, this determines which tags in the text are legal or illegal. The enumerated type `PeterBlum.DES.Security.HTMLTagMode` has these values:
  - `AllIllegal_NoLTGT` - Do not allow any < or > characters, regardless of if they form tags.
  - `AllIllegal` - All tags found are illegal.
  - `AllLegal` - All tags found are legal unless they are defined in the `<illegaltags>` section of the Peter's Input Security configuration files
  - `LegalExceptTags` - All tags are legal except those defined in the **HTMLTags** property and in the `<illegaltags>` section of the Peter's Input Security configuration files.
  - `IllegalExceptTags` - All tags are illegal except those in the **HTMLTags** property.

    This is the default setting for this property.

- **HTMLTags** (string) – When detecting script injection attacks, this can contain a list of HTML tag names. The **HTMLTagMode** determines how to use these tags.

  Always omit the < and > characters. Separate each tag with semicolons. Do not include spaces. For example, when looking for `<a>`, `<br>` and `<img>` use "`a;br;img`".

  Case insensitive testing is performed.

- **Enabled** (boolean) – When `true`, the ValidationAttribute is used. It defaults to `true`. Generally you set this to `false` using the `CustomizeColumn()` method when the ValidationAttribute was defined in metadata but will not be used on the current row.

- **DisplayOrder** (int) – When multiple ValidationAttributes are used on a property, use this to determine the order they are shown both inline and in the ValidationSummary. The higher the value, the lower it appears. When not specified, it uses 0. If multiple ValidationAttributes have the same **DisplayOrder**, they are shown in the opposite order they are listed in the metadata.

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

Example

[C#]

```
[DES.InjectionSecurity(SQLDetectionLevel=
  PeterBlum.DES.Security.SQLDetectionLevel.Medium, HTMLTagMode=AllLegal)]
```

[VB]

```
<DES.InjectionSecurity(SQLDetectionLevel:= _
  PeterBlum.DES.Security.SQLDetectionLevel.Medium, HTMLTagMode:=AllLegal)> _
```

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

DES.CharacterSetAttribute

Class	PeterBlum.DES.DynamicData.CharacterSetAttribute
Validation	CharacterValidator
Default ErrorMessage	`"The field {LABEL} contains an illegal character."`

Using this Attribute

Use on string columns that restrict the character set. On Field Templates that use the FilteredTextBox for data entry, this attribute also sets up the character set rules of the FilteredTextBox.

If this validation rule can be enabled and disabled based on some condition, such as when another column has a value, also use a DependencyAttribute on this column's metadata. See "DependencyAttributes: Relationships between columns".

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn".

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the Validator control. See "ErrorMessages on ValidationAttributes".
- **LettersLowercase** (Boolean) – When `true`, all lowercase letters are part of the characterset. When `false`, no lowercase letters are included unless they are specified in the **OtherCharacters** property. It defaults to `false`.
- **LettersUppercase** (Boolean) – When `true`, all uppercase letters are part of the characterset. When `false`, no uppercase letters are included unless they are specified in the **OtherCharacters** property. It defaults to `false`.
- **DiacriticLetters** (Boolean) – When `true`, diacritic (accented) letters are part of the characterset. There are two groups of diacritics available to users: ASCII and Unicode. They are both supported.

  When `false`, no diacritic letters are included unless they are specified in the **OtherCharacters** property. It defaults to `false`.

  #### ASCII Diacritic Letters Supported

  `âäàåáÄÅ çÇ éêëèÉ æÆ ƒ ïîìí ñÑ ôöòóÖ üûùúÜ ÿ`

  These letters are found in the ASCII character set between ASCII 128 and 165. Users on Windows type `ALT+1##` to insert them into the textbox. Users on MacOSX use **Edit; Special Characters** to open the Character Palette. These characters are in the Glyph View.

  #### Unicode Diacritic Letters Supported

  `àáâãäåÀÁÂÃÄÅ æÆ çÇ èéêëÈÉÊË ìíîïÌÍÎÏ ñÑ òóôõöøÒÓÔÕÖØ ùúûüÙÚÛÜ ýÿÝ ðÐ Þþ ß`

  These letters are found in Unicode's Latin-1 Supplement character set between 0192 (Hex 00C0) and 0255 (Hex 00FF). Users on Windows type `ALT+0###` to insert them into the textbox. Users on MacOSX use **Edit; Special Characters** to open the Character Palette. These characters are in the Unicode View.

- **Digits** (Boolean) – When `true`, all digits are part of the characterset. When `false`, no digits are included unless they are specified in the **OtherCharacters** property. It defaults to `false`.
- **Space** (Boolean) – When `true`, the Space character is part of the characterset. It defaults to `false`. The **Trim** property automatically strips leading and trailing spaces when `true`. It will not affect embedded spaces.
- **Enter** (Boolean) – When `true`, the Enter character is part of the characterset. It defaults to `false`.
- **Punctuation** (Boolean) – When `true`, punctuation characters are part of the characterset. When `false`, no punctuation characters are included unless they are specified in the **OtherCharacters** property. It defaults to `false`.

  #### Punctuation Characters Supported

  `. , ! ? ' " - ; :`

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

(period, comma, exclamation point, question mark, single quote, double quote, dash, semicolon, colon)

ALERT: Hackers use the single quote and dash for SQL Injection attacks. If you permit these characters, see the DES: Peter's Input Security User's Guide for defensive measures.

- **CurrencySymbols** (Boolean) – When `true`, currency symbol characters are part of the characterset. When `false`, no currency symbol characters are included unless they are specified in the **OtherCharacters** property. It defaults to `false`.

Currency Symbols Supported

`$ ¢ ¥ ¥ £ ¤`

(US Dollar, cents – UNICODE 0162, Yen – ASCII 157, Yen – UNICODE 0164, Pounds – UNICODE 0163)

- **EnclosureSymbols** (Boolean) – When `true`, characters that enclose (bracket) text are part of the characterset. When `false`, no enclosure characters are included unless they are specified in the **OtherCharacters** property. It defaults to `false`.

Enclosure Symbols Supported

`( ) [ ] { }`

- **MathSymbols** (Boolean) – When `true`, math symbol characters are part of the characterset. When `false`, no math symbol characters are included unless they are specified in the **OtherCharacters** property. It defaults to `false`.

Math Symbols Supported

`+ - * / = ( ) < > . % ± × ÷ ± ÷`

(plus, minus, asterisk, equals, left paren, right paren, less than, greater than, period, percent, plus/minus – UNICODE 0177, multiply – UNICODE 0215, divide – UNICODE - 0247, divide – ASCII 246)

ALERT: Hackers use the dash for SQL Injection attacks. If you permit the dash, see the DES: Peter's Input Security User's Guide for defensive measures.

- **VariousSymbols** (Boolean) – When `true`, various symbol characters shown below are part of the characterset. When `false`, none of the characters shown below are included unless they are specified in the **OtherCharacters** property. It defaults to `false`.

Various Symbols Supported

`_ @ # ^ & * ~ ¿ ¿ ¡ \ / | ¦ § © ® ` ´`

(Underscore, at, circumflex, ampersand, asterisk, tilde, inverted question mark – ASCII 168, inverted question mark – UNICODE 0191, inverted exclamation point – UNICODE 0161, left slash, right slash, pipe, broken bar, section sign, copyright, registered, grave accent, acute accent)

- **OtherCharacters** (string) – Enter each unique character that you want in the character. Don't enter any characters that are already covered by the previously stated properties. Suppose you want to support only "n", "x", and the underscore character here. You would enter "`nx_`". Suppose you want some common punctuation characters. You would enter "`.!,?;:`".

- **Exclude** (Boolean) – When `true`, the character set defined in the above properties is excluded; all other characters are permitted. When `false`, the character set defined above are the only characters permitted. It defaults to `false`.

- **Enabled** (boolean) – When `true`, the ValidationAttribute is used. It defaults to `true`. Generally you set this to `false` using the `CustomizeColumn()` method when the ValidationAttribute was defined in metadata but will not be used on the current row.

- **DisplayOrder** (int) – When multiple ValidationAttributes are used on a property, use this to determine the order they are shown both inline and in the ValidationSummary. The higher the value, the lower it appears. When not specified, it uses 0. If multiple ValidationAttributes have the same **DisplayOrder**, they are shown in the opposite order they are listed in the metadata.

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

Example: Only letters, Period and Dash

[C#]

```
[DES.CharacterSet(LettersUppercase=true, LettersLowercase=true,
  OtherCharacters=".-")]
```

[VB]

```
<DES.CharacterSet(LettersUppercase:=True, LettersLowercase:=True, _
  OtherCharacters:=".-")> _
```

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

DES.CreditCardNumberAttribute

Class	PeterBlum.DES.DynamicData.CreditCardNumberAttribute
Validation	CreditCardNumberValidator
Default ErrorMessage	`"The field {LABEL} not a valid credit card number."`

Using this Attribute

Use when the string contains a credit card number.

The CreditCardNumberAttribute also serves as a DataTypeAttribute with the ExtendedDataType=CreditCardNumber. As a result, if you have a Field Template with these names, they will be used: CreditCardNumber.ascx, CreditCardNumber_Edit.ascx, and CreditCardNumber_Insert.ascx. If you use the CustomizeColumn() method to create the CreditCardNumberAttribute, it will be created too late to select the CreditCardNumber Field Template. In that case, assign the DataTypeAttribute(ExtendedDataType.CreditCardNumber) to the column's metadata and create the CreditCardNumberAttribute to supply the Field Template with its CreditCardNumberValidator.

If this validation rule can be enabled and disabled based on some condition, such as when another column has a value, also use a DependencyAttribute on this column's metadata. See "DependencyAttributes: Relationships between columns".

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn".

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the Validator control. See "ErrorMessages on ValidationAttributes".
- **CheckByBrand** (Boolean) – Determines if the credit card brands are checked. When `true`, they are.

  Credit card brands are defined in the **des.config** and **custom.des.config** files, in the `<CreditCards>` section. You can modify the **custom.des.config** file to add new cards or remove those defined in the **des.config** file (the factory defaults.) See "Editing Credit Card Brands" in the **Validation User's Guide**.

  It defaults to `true`.
- **IgnoreBlankText** (Boolean) – Determines how blank text is evaluated. When `true`, the Validator cannot be evaluated. When `false`, the Validator evaluates as failed (reports an error). It defaults to `true`.
- **AllowSpaces** (Boolean) – You may prefer to let users enter spaces in their credit card number. Your server side code will strip out those spaces for its own use. If this is the case, set **AllowSpaces** to `true`. The validator does not care where the spaces are located or if there are several in a row.

  It defaults to `false`.
- **Enabled** (boolean) – When `true`, the ValidationAttribute is used. It defaults to `true`. Generally you set this to `false` using the `CustomizeColumn()` method when the ValidationAttribute was defined in metadata but will not be used on the current row.
- **DisplayOrder** (int) – When multiple ValidationAttributes are used on a property, use this to determine the order they are shown both inline and in the ValidationSummary. The higher the value, the lower it appears. When not specified, it uses 0. If multiple ValidationAttributes have the same **DisplayOrder**, they are shown in the opposite order they are listed in the metadata.

Example

[C#]

```
[DES.CreditCardNumber()]
```

[VB]

```
<DES.CreditCardNumber()> _
```

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

DES.ABARoutingNumberAttribute

Class	PeterBlum.DES.DynamicData.ABARoutingNumbersAttribute
Validation	ABARoutingNumberValidator
Default ErrorMessage	`"The field {LABEL} not a valid routing number."`

Using this Attribute

Use when the string contains an ABA Routing Number.

The ABARoutingNumberAttribute also serves as a DataTypeAttribute with the ExtendedDataType=ABARoutingNumber. As a result, if you have a Field Template with these names, they will be used: ABARoutingNumber.ascx, ABARoutingNumber_Edit.ascx, and ABARoutingNumber_Insert.ascx. If you use the CustomizeColumn() method to create the ABARoutingNumberAttribute, it will be created too late to select the ABARoutingNumber Field Template. In that case, assign the DataTypeAttribute(ExtendedDataType.ABARoutingNumber) to the column's metadata and create the ABARoutingNumberAttribute to supply the Field Template with its ABARoutingNumberValidator.

If this validation rule can be enabled and disabled based on some condition, such as when another column has a value, also use a DependencyAttribute on this column's metadata. See "DependencyAttributes: Relationships between columns".

Properties

These properties can also be established as parameters on the metadata. When setting it programmatically, see "Using the CustomizeColumn".

- **ErrorMessage**, **ErrorMessageLookupID**, **SummaryErrorMessage**, **SummaryErrorMessageLookupID** (string) – Corresponds to the same properties on the Validator control. See "ErrorMessages on ValidationAttributes".
- **IgnoreBlankText** (Boolean) – Determines how blank text is evaluated. When `true`, the Validator cannot be evaluated. When `false`, the Validator evaluates as failed (reports an error). It defaults to `true`.
- **Enabled** (boolean) – When `true`, the ValidationAttribute is used. It defaults to `true`. Generally you set this to `false` using the CustomizeColumn() method when the ValidationAttribute was defined in metadata but will not be used on the current row.
- **DisplayOrder** (int) – When multiple ValidationAttributes are used on a property, use this to determine the order they are shown both inline and in the ValidationSummary. The higher the value, the lower it appears. When not specified, it uses 0. If multiple ValidationAttributes have the same **DisplayOrder**, they are shown in the opposite order they are listed in the metadata.

Example

[C#]

```
[DES.ABARoutingNumber()]
```

[VB]

```
<DES.ABARoutingNumber()> _
```

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

ValidationAttributes that define the DataTypeCheckValidator

Some ValidationAttributes also work as a DataTypeAttribute. The DataTypeAttribute provides a more specialized datatype than the column specifies. It also defines a different default Field Template. See "DataTypeAttributes: The DataTypes used by your Business logic". Those ValidationAttributes that work as a DataTypeAttribute add the ability to define a Validation rule on the column's metadata.

A natural case where DataType and Validation are merged is with a DataTypeCheckValidator. This validator needs to know a specific datatype. So use one of these ValidationAttributes to define both the datatype and its validation rule.

Click on the Attribute name to jump to its description and properties.

DataType	Attribute	Field Template used
Integer	DES.IntegerDataTypeAttribute	Integer
Decimal	DES.DecimalDataTypeAttribute	Decimal
Currency	DES.CurrencyDataTypeAttribute	Currency
Percent	DES.PercentDataTypeAttribute	Percent
Integer used as a measurement	DES.IntegerMeasurementDataTypeAttribute	Measurement
Decimal used as a measurement	DES.DecimalMeasurementDataTypeAttribute	Measurement
Date	DES.DateDataTypeAttribute	Date
Time of day	DES.TimeOfDayDataTypeAttribute	Time
Duration	DES.DurationDataTypeAttribute	Duration
Date without year	DES.AnniversaryDataTypeAttribute	Anniversary
Date without day of month	DES.MonthYearDataTypeAttribute	MonthYear

Note: If you have a DateTime value that should represent both date and time of day, you do not need any DataTypeAttribute as the default Field Templates for DateTime are used and know how to setup a DataTypeCheckValidator for this situation.

If you use the `CustomizeColumn()` method to create the one of these ValidationAttributes, it will be created too late to select the default Field Template shown above. In that case, assign the DataTypeAttribute to the column's metadata and create the ValidationAttribute to supply the Field Template with its DataTypeCheckValidator.

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

DependencyAttributes: Relationships between columns

Often one column can be ignored or disabled based on the value of another column. Say for example you have a Mode column which takes an integer with 3 possible values: 0, 1, and 2. When the Mode=2, the ModeDescription column requires a string value. When Mode=0 or 1, ModeDescription is not used. In this case, you want the ModeDescription to have a RequiredAttribute supplying a RequiredTextValidator. Yet that validator should be disabled depending on the mode. DES's validators have the Enabler property where you specify a rule to enable and disable the validator.

DES Dynamic Data provides the DependencyAttribute to:

- Establish the Enabler property's rule on any validators assigned to the column.
- Optionally disable or hide the data entry control.

The `PeterBlum.DES.DynamicData.BaseDependencyAttribute` class is used to develop DependencyAttribute classes. DES Dynamic Data comes with these DependencyAttributes:

- **RequiredDependencyAttribute** – Evaluates one or more columns to determine if they have been assigned a value. See "RequiredDependencyAttribute".
- **ConditionDependencyAttribute** – Use any of DES's Condition classes, giving you extraordinary flexibility to evaluate values on other columns. See the **Validation User's Guide** for all available Condition classes.

Click on any of these topics to jump to them:

- Using DependencyAttributes
- Using the Column's Metadata
- Using the CustomizeColumn Method
- Using the CustomizeColumn
- RequiredDependencyAttribute
- ConditionDependencyAttribute

Using DependencyAttributes

Always assign the DependencyAttribute to the column that you need to enable and disable its validators and optionally data entry control. The DependencyAttributes have properties where you assign the names of the others columns that the column of interest depends on.

You can establish DependencyAttributes on the column's metadata or programmatically using the `CustomizeColumn()` method as described in the next two sections.

A DependencyAttribute establishes the **Enabler** property on each validator of the column. If the Validator already has its Enabler defined, the DependencyAttribute uses the **ReplaceEnabler** property to determine if its condition replaces the existing one, does not replace, or merges with the other using an AND or OR operator.

The data entry control of the Field Template that uses the DependencyAttribute can have a different visual appearance when the Dependency indicates the column is disabled. By default, the data entry control is disabled. (The HTML tag gets its disabled property assigned.) If you add the Customizer control, use its **DependencyUI** property to select between these interfaces: none, disable, and invisible.

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

Using the Column's Metadata

1. Open the code file containing the EntityTable Metadata classes in an editor.
2. Go to the column (property) within the EntityTable Metadata class that needs a DependencyAttribute.
3. Apply the attribute.

Example: RequiredDependencyAttribute

Require at least one of the columns PrinterType and FaxType have data. See "RequiredDependencyAttribute" for details.

[C#]

```
[DES.RequiredDependency(ColumnNames="PrinterType|FaxType",
  MultipleMode=PeterBlum.DES.MultipleRequiredControlsMode.AtLeastOne)]
```

[VB]

```
<DES.RequiredDependency(ColumnNames:="PrinterType|FaxType", _
  MultipleMode:=PeterBlum.DES.MultipleRequiredControlsMode.AtLeastOne) > _
```

Example: ConditionDependencyAttribute

The value of the Mode column must be equal to 2. It uses the CompareToValueCondition with Operator=Equals and ValueToCompare=2. See "ConditionDependencyAttribute" for details.

[C#]

```
[DES.ConditionDependency(Condition=typeof(PeterBlum.DES.CompareToValueCondition),
  PropertyList="Operator='PeterBlum.DES.ConditionOperator.Equals' " +
     "ValueToCompare='2'"]
```

[VB]

```
<[DES.ConditionDependency(Condition:=GetType(PeterBlum.DES.CompareToValueCondition), _
  PropertyList:="Operator='PeterBlum.DES.ConditionOperator.Equals' " +
      "ValueToCompare='2'" > _
```

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

Using the CustomizeColumn Method

You can programmatically define or edit a DependencyAttribute by implementing the `PeterBlum.DES.DynamicData.ICustomizeColumn` interface on your EntityTable or the `PeterBlum.DES.DynamicData.IUICustomizeColumn` interface on your web form. See "Customizing Metadata At Runtime". Both define the `CustomizeColumn()` method.

The `CustomizeColumn()` method is passed a `PeterBlum.DES.DynamicData.ModifiableColumn` object. Its **DependencyAttribute** property contains the current DependencyAttribute. You can edit it or replace it. These methods are available to make editing it easier:

- `GetDependencyAttribute()` – Retrieve the current DependencyAttribute. If none exists, it returns `null`. Same as getting the value of the **ModifiableColumn.DependencyAttribute** property. See "Method: GetDependencyAttribute".
- `CreateRequiredDependencyAttribute()` – Creates a new RequiredDependencyAttribute for use with this column. See "Method: CreateRequiredDependencyAttribute".
- `CreateConditionDependencyAttribute()` – Creates a new ConditionDependencyAttribute for use with this column. See "Method: CreateConditionDependencyAttribute".
- `RemoveDependencyAttribute()` – Eliminates a DependencyAttribute already on this column. See "Method: RemoveDependencyAttribute".

Note: All of these methods only affect the current Field Template's request for a DependencyAttribute. It does not affect the global metadata.

Method: GetDependencyAttribute

Retrieve the current DependencyAttribute. If none exists, it returns `null`. Remember to typecast the result.

Use it to access the metadata's DependencyAttribute to modify its properties. If you want to eliminate the DependencyAttribute from the metadata, use `RemoveDependencyAttribute()`.

Your edits to the result object do not affect the global metadata for this column.

[C#]

```
public BaseDependencyAttribute GetDependencyAttribute()
```

[VB]

```
Public Function GetDependencyAttribute() As BaseDependencyAttribute
```

Returns

Retrieves the current DependencyAttribute. If none exists, it returns `null`.

SEE EXAMPLE ON THE NEXT PAGE

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

Example

Change the Condition object on the existing ConditionDependencyAttribute of the PostalCode field of the Customer table.

[C#]

```
[MetadataType(typeof(CustomerMetadata))]
public partial class Customer : DES.ICustomizeColumn
{
   public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn, bool pIsNewRow)
   {
      switch (pModifiableColumn.ColumnName)
      {
         case "PostalCode":
            DES.ConditionDependencyAttribute vCDA =
              (DES.ConditionDependencyAttribute)
                   pModifiableColumn.GetDependencyAttribute();
            vCDA.Condition = new PeterBlum.DES.TextLengthCondition(null, "City", 10);
            break;
      }  // switch
   }
}
public class CustomerMetadata
{
   [DES.ConditionDependency(typeof(PeterBlum.DES.RequiredTextCondition),
     "ControlIDToEvaluate='City'")]
   public object PostalCode { get; set; }
}
```

[VB]

```
<MetadataType(GetType(CustomerMetadata))> _
Partial Public Class Customer _
     Implements DES.ICustomizeColumn

   Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn,_
     ByVal pIsNewRow As Boolean)

      Select Case pModifiableColumn.ColumnName
         Case "PostalCode"
            Dim vCDA As DES.ConditionDependencyAttribute = _
              CType(pModifiableColumn.GetDependencyAttribute(), _
                DES.ConditionDependencyAttribute)
            vCDA.Condition = New PeterBlum.DES.TextLengthCondition(Nothing, "City", 10)
      End Select
   End Sub
End Class
Public Class CustomerMetadata
   <DES.ConditionDependency(GetType(PeterBlum.DES.RequiredTextCondition), _
     "ControlIDToEvaluate='City'"> _
   Public Property PostalCode As Object
     Get
       Return Nothing
     End Get
     Set
     End Set
   End Property
End Class
```

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

Method: CreateRequiredDependencyAttribute

Create a RequiredDependencyAttribute to be used by the requesting Field Template. (It does not affect the global metadata for this column.) If the metadata already has a DependencyAttribute, it is replaced by the one created.

The parameters will setup the **ColumnNames** and **MultipleMode** properties on the RequiredDependencyAttribute. To modify anything else, use the result from this function as it contains the RequiredDependencyAttribute.

[C#]

```
public RequiredDependencyAttribute CreateRequiredDependencyAttribute(
   string pColumnNames,
   MultipleRequiredControlsMode pMultipleMode)
```

[VB]

```
Public Function CreateRequiredDependencyAttribute( _
   ByVal pColumnNames As String, _
   ByVal pMultipleMode As MultipleRequiredControlsMode) As RequiredDependencyAttribute
```

Parameters

pColumnNames

A pipe (|) delimited list of one or more column names that are required for this to be enabled. For example: "City|State".

pMultipleMode

When multiple columns have been defined, this determines how many of the columns must have data to enable the column that owns this DependencyAttribute. If there is only one column name, use `PeterBlum.DES.MultipleRequiredControlsMode.All`.

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

- `All` – All must have data.
- `AllOrNone` – Either all must have data or none have data.
- `AtLeastOne` – At least one must have data.
- `OnlyOne` – Only one can have data.
- `Range` – This is not supported here.

Example

Make the Address2 column dependent on the Address1 column. No metadata has been defined on the Address2 column.

[C#]

```
[MetadataType(typeof(CustomerMetadata))]
public partial class Customer : DES.ICustomizeColumn
{
   public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn, bool pIsNewRow)
   {
      switch (pModifiableColumn.ColumnName)
      {
         case "Address2":
            pModifiableColumn.CreateRequiredDependencyAttribute("Address1",
              PeterBlum.DES.MultipleRequiredControlsMode.All);
            break;
      }  // switch
   }
}
```

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

[VB]

```
<MetadataType(GetType(CustomerMetadata))> _
Partial Public Class Customer _
     Implements DES.ICustomizeColumn

   Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn,_
     ByVal pIsNewRow As Boolean)

      Select Case pModifiableColumn.ColumnName
         Case "Address2"
            pModifiableColumn.CreateRequiredDependencyAttribute("Address1", _
              PeterBlum.DES.MultipleRequiredControlsMode.All)
      End Select
   End Sub
End Class
```

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

Method: CreateConditionDependencyAttribute

Create a ConditionDependencyAttribute to be used by the requesting Field Template. (It does not affect the global metadata for this column.) If the metadata already has a DependencyAttribute, it is replaced by the one created.

The parameter will setup the **Condition** property on the ConditionDependencyAttribute. To modify anything else, use the result from this function as it contains the ConditionDependencyAttribute.

[C#]

```
public ConditionDependencyAttribute CreateConditionDependencyAttribute(
   BaseCondition pCondition)
```

[VB]

```
Public Function CreateConditionDependencyAttribute( _
   ByVal pCondition As BaseCondition) As ConditionDependencyAttribute
```

Parameters

pCondition

The Condition to be used by the ConditionDependencyAttribute.

Always assign column names to the **ControlIDToEvaluate** and **SecondControlIDToEvaluate** properties. (Do not use the DisplayNameAttribute value for column names. Use the original name.)

Always pass `null`/`Nothing` for the Owner parameter of any Condition's constructor.

Example

The PostalCode column depends on the length of the City column being no more than 10 characters.

[C#]

```
[MetadataType(typeof(CustomerMetadata))]
public partial class Customer : DES.ICustomizeColumn
{
   public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn, bool pIsNewRow)
   {
      switch (pModifiableColumn.ColumnName)
      {
         case "PostalCode":
            pModifiableColumn.CreateConditionDependencyAttribute(
               new PeterBlum.DES.TextLengthCondition(null, "City", 10));
            break;
      }  // switch
   }
}
```

[VB]

```
<MetadataType(GetType(CustomerMetadata))> _
Partial Public Class Customer _
     Implements DES.ICustomizeColumn

   Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn,_
     ByVal pIsNewRow As Boolean)

      Select Case pModifiableColumn.ColumnName
         Case "Address2"
            pModifiableColumn.CreateConditionDependencyAttribute( _
               New PeterBlum.DES.TextLengthCondition(Nothing, "City", 10))
      End Select
   End Sub
End Class
```

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

Method: RemoveDependencyAttribute

Remove the DependencyAttribute set by metadata. (It does not affect the global metadata for this column.)

[C#]

```
public void RemoveDependencyAttribute()
```

[VB]

```
Public Sub RemoveDependencyAttribute()
```

Example

Remove the existing ConditionDependencyAttribute on the PostalCode field of the Customer table.

[C#]

```
[MetadataType(typeof(CustomerMetadata))]
public partial class Customer : DES.ICustomizeColumn
{
   public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn, bool pIsNewRow)
   {
      switch (pModifiableColumn.ColumnName)
      {
         case "PostalCode":
            pModifiableColumn.RemoveDependencyAttribute();
            break;
      }  // switch
   }
}
public class CustomerMetadata
{
[DES.ConditionDependency(typeof(PeterBlum.DES.RequiredTextCondition),
     "ControlIDToEvaluate='City'")]
  public object PostalCode { get; set; }
}
```

[VB]

```
<MetadataType(GetType(CustomerMetadata))> _
Partial Public Class Customer _
     Implements DES.ICustomizeColumn

   Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn,_
     ByVal pIsNewRow As Boolean)

      Select Case pModifiableColumn.ColumnName
         Case "PostalCode"
            pModifiableColumn.RemoveDependencyAttribute()
      End Select
   End Sub
End Class
Public Class CustomerMetadata
  <DES.ConditionDependency(GetType(PeterBlum.DES.RequiredTextCondition), _
     "ControlIDToEvaluate='City'"> _
  Public Property PostalCode As Object
     Get
       Return Nothing
     End Get
     Set
     End Set
  End Property
End Class
```

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

RequiredDependencyAttribute

This DependencyAttribute enables the column based on whether other columns have been assigned a value or not. It establishes the Enabler property on Validators. It optionally disables or hides the data entry control on the Field Template.

When assigning the attribute as metadata, it looks like this:

[C#]

```
[DES.RequiredDependency("columnname")]
```

```
[DES.RequiredDependency(ColumnNames="columnname")]
```

```
[DES.RequiredDependency("columnname|columnname",
  PeterBlum.DES.MultipleRequiredControlsMode.AtLeastOne)]
```

```
[DES.RequiredDependency(ColumnNames="columnname|columnname"),
  MultipleMode=PeterBlum.DES.MultipleRequiredControlsMode.AtLeastOne)]
```

[VB]

```
<DES.RequiredDependency("columnname")> _
```

```
<DES.RequiredDependency(ColumnNames:="columnname")> _
```

```
<DES.RequiredDependency("columnname|columnname", _
  PeterBlum.DES.MultipleRequiredControlsMode.AtLeastOne) > _
```

```
<DES.RequiredDependency(ColumnNames="columnname|columnname"), _
  MultipleMode=PeterBlum.DES.MultipleRequiredControlsMode.AtLeastOne) > _
```

When assigning it programmatically, see “Method: CreateConditionDependencyAttribute”.

Properties

These properties can also be established as parameters on the metadata.

- **ColumnNames** (string) – A pipe (`|`) delimited list of one or more column names that are required for this to be enabled. For example, “City|State”.
- **MultipleMode** (enum PeterBlum.DES.MultipleRequiredControlsMode) – When multiple columns have been defined in the **ColumnNames** property, this determines how many of the columns must have data to enable the column that owns this DependencyAttribute.

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

  - `All` – All must have data. This is the default.
  - `AllOrNone` – Either all must have data or none have data.
  - `AtLeastOne` – At least one must have data.
  - `OnlyOne` – Only one can have data.
  - `Range` – This is not supported here.
- **UnassignedValues** (string) – A semicolon (`;`) delimited list of the unassigned values by column. Each element corresponds to an item in the **ColumnNames** property.

  There must be the same number of elements or exactly one. When there is exactly one and the **ColumnNames** has multiple values, the same element is applied to all columns.

  When "", all columns use "" for their unassigned value.

  Each element can contain multiple unassigned values by using a pipe delimited list. For example: `"this|that;those|these"`

  When evaluating a checkbox, use "" when unchecked is unassigned and "checked" (actually any text) when checked is unassigned.

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

- **UnassignedCaseInsensitive** (boolean) – Determines if the **UnassignedValues** property uses a case insensitive match or not. When `true`, its case insensitive. It defaults to `true`.
- **ReplaceEnabler** (enum PeterBlum.DES.DynamicData.ReplaceEnabler) – When the Field Template contains validators whose Enabler properties are setup, this determines how the DependencyAttribute modifies those Enabler properties.

  The enumerated type `PeterBlum.DES.DynamicData.ReplaceEnabler` has these values:

  - `No` – Do not replace. The default
  - `Yes` – Always replace
  - `MultiCondition_OR` – Merge with the existing Condition. The two conditions will be evaluated with an OR condition so that if either evaluates as "success", the Dependency enables the validators and data entry control.
  - `MultiCondition_AND` – Merge with the existing Condition. The two conditions will be evaluated with an AND condition so that when both evaluate as "success", the Dependency enables the validators and data entry control.

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

ConditionDependencyAttribute

This DependencyAttribute enables the column based on your choice of Condition object. Condition classes are the logic within validators that determines validity. See the "Using Conditions" section of the **Validation User's Guide**. It establishes the Enabler property on Validators. It optionally disables or hides the data entry control on the Field Template.

When assigning the attribute as metadata, it looks like this:

[C#]

```
[DES.ConditionDependency(typeof(PeterBlum.DES.Condition),
   "propertyname='value' propertyname='value'")]
```

```
[DES.ConditionDependency(TypeOfCondition=typeof(PeterBlum.DES.Condition)),
   PropertyList="propertyname='value' propertyname='value'")]
```

[VB]

```
<DES.ConditionDependency(GetType(PeterBlum.DES.Condition),
   "propertyname='value' propertyname='value'")> _
```

```
<DES.ConditionDependency(TypeOfCondition=GetType(PeterBlum.DES.Condition)), _
   PropertyList="propertyname='value' propertyname='value'") > _
```

The parameters differ from the properties shown below. The parameters work together to create and populate properties on a Condition object.

- **TypeOfCondition** (Type) – The class type of the Condition.
- **PropertyList** (string) – Names and values of properties to set on the Condition. The syntax is the same as within a web control's properties, except that each value must be surrounded by single quotes.

  ```
  "[propertyname]='value'[space][additional property definitions]"
  ```

  Values start as strings and use the TypeConverter class associated with the property's data type to convert to a real value. So the strings must be in a format that the TypeConverter recognizes. Some examples:

  - Integers – digits; lead minus as the negative character
  - Decimals – digits; period as the decimal separator; lead minus as the negative character
  - Dates – yyyy-MM-dd format (culture neutral formatting)
  - Times and durations – hh:mm:ss format
  - Enumerated – The enumerated value without the namespace or type
  - Strings – Since the string is enclosed in single quotes, use the XML symbol `'` for a single quote (apostrophe). Example: `Description='Peter's Data Entry Suite'`

  Example

  Setting the CompareToValueCondition with ValueToCompare=10 and Operator=LessThan.

  [C#]

  ```
  [DES.ConditionDependency(typeof(PeterBlum.DES.CompareToValueCondition),
   "ValueToCompare='10' Operator='LessThan'")]
  ```

  [VB]

  ```
  <DES.ConditionDependency(GetType(PeterBlum.DES.CompareToValueCondition), _
   "ValueToCompare='10' Operator='LessThan'")> _
  ```

When assigning it programmatically, see "Method: CreateConditionDependencyAttribute".

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

Properties

These properties can also be established as parameters on the metadata.

- **Condition** (subclass of PeterBlum.DES.BaseCondition) – The Condition object to be assigned to the Enabler property in the validators.

  Always assign column names to the **ControlIDToEvaluate** and **SecondControlIDToEvaluate** properties. (Do not use the DisplayNameAttribute value for column names. Use the original name.)

  Always pass `null`/`Nothing` for the Owner parameter of any Condition's constructor.

- **ReplaceEnabler** (enum PeterBlum.DES.DynamicData.ReplaceEnabler) – When the Field Template contains validators whose Enabler properties are setup, this determines how the DependencyAttribute modifies those Enabler properties.

  The enumerated type `PeterBlum.DES.DynamicData.ReplaceEnabler` has these values:

  - `No` – Do not replace. The default
  - `Yes` – Always replace
  - `MultiCondition_OR` – Merge with the existing Condition. The two conditions will be evaluated with an OR condition so that if either evaluates as "success", the Dependency enables the validators and data entry control.
  - `MultiCondition_AND` – Merge with the existing Condition. The two conditions will be evaluated with an AND condition so that when both evaluate as "success", the Dependency enables the validators and data entry control.

  This property can be passed as a parameter to the ConditionDependencyAttribute metadata as "ReplaceEnabler".

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

UIHintAttribute: Customizing the Field Template used

The UIHintAttribute can override which Field Template is selected. In addition, it can set properties on the Field Template.

Recommendation: When you are concerned with the separation between business logic and user interface, try to avoid using the UIHintAttribute. It overlaps the two areas. Business logic developers should use various DataTypeAttributes to select a default Field Template for a column and web form developers can override it with the DynamicField and DynamicControl controls.

There are actually two of these attributes, `System.ComponentModel.DataAnnotations.UIHintAttribute` comes with the native framework and `PeterBlum.DES.DynamicData.UIHintAttribute` comes with DES Dynamic Data. DES's attribute is a subclass of the native attribute, adding support for mode-specific Field Template names and property lists. Use either one, although if you need mode-specific features, use the one from DES.

By "mode-specific", the UIHintAttribute provides a different value depending on the mode of the DataBound control: ReadOnly, Edit, and Insert. This allows for different Field Templates based on the mode.

Selecting an alternative Field Template

You can develop more than one Field Template that supports a data type. For example, a double value is supported by Decimal.ascx, Currency.ascx, Percent.ascx, Duration.ascx, and Measurement.ascx. By using a DataTypeAttribute on the column's metadata, it loads the more specific Field Template, such as PercentDataTypeAttribute selects Percent.ascx. Yet that level of selectability is not enough. Suppose you have two ways to display or edit Percentage. In that case, you create an additional Field Template for the alternative interface and give it a different name. Use the UIHintAttribute to specify the name of the alternative Field Template.

The FieldTemplateFactory class, which is assigned to the MetaModel in `Application_Start()`, makes the decisions on which Field Template to use. If it cannot find the right template for a UIHintAttribute or DataTypeAttribute, it has fallback rules to ensure that some kind of template is provided. See "Understanding how Field Templates are selected".

The UIHint can actually be defined in several places:

- UIHintAttribute on the column's metadata, as shown below.
- **UIHint** property on a DynamicField object of a GridView and DetailsView control. If a UIHintAttribute is also on the metadata, the **UIHint** property takes precedence.
- **UIHint** property on a DataFieldInPattern object of a DynamicPattern, DynamicListView and DynamicFormView control. If a UIHintAttribute is also on the metadata, the **UIHint** property takes precedence.
- **UIHint** property on a DynamicControl. If a UIHintAttribute is also on the metadata, the **UIHint** property takes precedence.

Applying the UIHintAttribute to select an alternative Field Template

Here is the syntax for this attribute.

The native UIHintAttribute definition

[C#]

```
[UIHint("TemplateName")]
```

[VB]

```
<UIHint("TemplateName") > _
```

The DES UIHintAttribute definition

[C#]

```
[DES.UIHint("TemplateName")]
```

```
[DES.UIHint("TemplateName", ReadOnlyUIHint="TemplateName",
  EditUIHint="TemplateName", InsertUIHint="TemplateName")]
```

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

[VB]

```
<DES.UIHint("TemplateName")> _
```

```
<DES.UIHint("TemplateName", ReadOnlyUIHint:="TemplateName", _
  EditUIHint:="TemplateName", InsertUIHint:="TemplateName")> _
```

Where *TemplateName* does not include the "_edit", "_insert", or ".ascx" elements found on the file name.

TemplateName can be "" when using the UIHintAttribute to specify other properties but you want to use the standard Field Template for the given situation.

The properties on the DES.UIHintAttribute are:

- **ReadOnlyUIHint** (string) – The Field Template name requested for ReadOnly mode. If blank, use the UIHint property (the first parameter of the UIHintAttribute).
- **EditUIHint** (string) – The Field Template name requested for Edit mode. If blank, use the UIHint property (the first parameter of the UIHintAttribute).
- **InsertUIHint** (string) – The Field Template name requested for Insert mode. If blank, use the UIHint property (the first parameter of the UIHintAttribute).

Example: Using the Native UIHintAttribute

DES provides two Field Templates for date values, "Date" and "Calendar". (Calendar is only supplied for edit mode. If you select it for ReadOnly mode, Date.ascx will still be used.) This example still uses the DateDataTypeAttribute to specify the official type of the FinishDate column, which uses the DateTime type in the EntityTable class.

[C#]

```
[UIHint("Calendar")]
[DES.DateDataType()]
public object FinishDate { get; set; }
```

[VB]

```
<UIHint("Calendar"), _
<DES.DateDataType()]> _
Public Property FinishDate As Object
  Get
    Return Nothing
  End Get
  Set
  End Set
End Property
```

See also "How To: Customize Data Field Appearance and Behavior in a Dynamic Data Control"

Example: Using the DES UIHintAttribute with different modes

This is based on the previous example. It only uses the Calendar for Insert mode. The rest will use the standard Field Templates determined by the DateDataTypeAttribute.

[C#]

```
[DES.UIHint("", InsertUIHint="Calendar")]
[DES.DateDataType()]
public object FinishDate { get; set; }
```

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

[VB]

```
<DES.UIHint("", InsertUIHint:="Calendar")> _
<DES.DateDataType()]> _
Public Property FinishDate As Object
  Get
    Return Nothing
  End Get
  Set
  End Set
End Property
```

Setting properties on Field Templates

ALERT: This feature is only supported by the DES.UIHintAttribute, not the native one (System.ComponentModel.DataAnnotations.UIHintAttribute).

While you typically think of the UIHintAttribute to select a specific Field Template by name, it can deliver properties to the Field Template too. For example, every DES Field Template has the property **DataControlWidth** where you can specify the exact width of the control showing the data. For a list of properties on Field Templates, see "Field Template classes and properties".

The `PeterBlum.DES.DynamicData.UIHintAttribute` ("DES.UIHintAttribute") has several properties where you can define a property list for the Field Templates. They are **DefaultPropertyList**, **ReadOnlyPropertyList**, **EditPropertyList**, and **InsertPropertyList**. Since different Field Template files are used based on the mode, these properties allow you to define values based on the mode too. If you don't care about the mode, use **DefaultPropertyList**.

The format of these properties is a string containing ASP.NET markup, just as you would enter it into the control. Make sure you enclose each property in single quotes. Here's the overall pattern:

```
DefaultPropertyList="PropertyName1='propertyvalue' PropertyName2='propertyvalue'"
```

Hint: If you need single quotes within your property values, enter them as "&#39".

The DES.UIHintAttribute uses this syntax:

[C#]

```
[DES.UIHint("TemplateName", DefaultPropertyList="property list",
  ReadOnlyPropertyList="property list",
  EditPropetyList="property list", InsertPropertyList="property list")]
```

[VB]

```
<DES.UIHint("TemplateName", DefaultPropertyList="property list", _
  ReadOnlyPropertyList="property list", _
  EditPropetyList="property list", InsertPropertyList="property list")>_
```

Named parameters can be omitted when not needed.

The first parameter sets the template name. It can be "" when you want to let the Field Template to be selected by other rules. See "Applying the UIHintAttribute to select an alternative Field Template".

The remaining parameters are described here:

- **DefaultPropertyList** (string) – A property list that is always applied the Field Template. See below for the syntax. The mode-specific PropertyLists can override identical properties found here. Any unique properties declared here will still be set. This allows you to setup properties common to all modes here.
- **ReadOnlyPropertyList** (string) – A property list that is applied to a Field Template in ReadOnly mode. See below for the syntax. This property list is applied after DefaultPropertyList, allowing individual properties to override values already applied.
- **EditPropertyList** (string) – A property list that is applied to a Field Template in Edit mode. See below for the syntax. This property list is applied after DefaultPropertyList, allowing individual properties to override values already applied.

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

- **InsertPropertyList** (string) – A property list that is applied to a Field Template in Insert mode. See below for the syntax. This property list is applied after DefaultPropertyList, allowing individual properties to override values already applied.

PropertyList Syntax

The PropertyList properties use the ASP.NET Declarative Syntax, which means it looks like properties you would type into an `<asp:class here>` tag. However, you should enclose every value in single quotes. If you need to declare a single quote inside the value, use the text "&#39".

For example, `InsertPropertyList="DataControlWidth='40px' ValueWhenBlank='FillMe'"`.

The property value is a string that must be compatible with the TypeConverter for the property's datatype. For example, a DateTime property uses the DateTimeTypeConverter. It converts strings using the ShortDatePattern in the current thread's CultureInfo object or using the culture neutral format "yyyy-MM-dd".

You can refer to the child properties of properties using the syntax: `PropertyName-ChildPropertyName='value'`. This is important because you have access to the data control through the **DataControl** property and the ColumnValidatorManager through the **ColumnValidatorManager** property.

For example, `InsertPropertyList="DataControl-BackColor='aliceblue'"`.

When you declare an unknown property name, it will be ignored. If you want to see any errors caused by incorrect property names or values, set `<% @Page Trace="true" %>` and view the trace.

Examples

[C#]

```
[UIHint(DefaultPropertyList="DataControlWidth='100px'"
      InsertPropertyList="DataControlWidth='80px' ValueWhenBlank='FillMe'")]
[UIHint("Calendar", DefaultPropertyList="UseDescriptionAsToolTip='false'")]
[UIHint(InsertPropertyList="DataControl-BackColor='aliceblue'")]
```

[VB]

```
<UIHint(DefaultPropertyList="DataControlWidth='100px'" _
      InsertPropertyList="DataControlWidth='80px' ValueWhenBlank='FillMe'")> _
<UIHint("Calendar", DefaultPropertyList="UseDescriptionAsToolTip='false'")> _
<UIHint(InsertPropertyList="DataControl-BackColor='aliceblue'")> _
```

When creating DES.UIHintAttribute programmatically

You can create or modify a UIHintAttribute object programmatically by using the ICustomizableColumn and IUICustomizableColumn interfaces. In this case, don't assign values to the above properties because those properties require parsing and conversion to their native types. That uses extra CPU time that can be avoided by using the following technique.

The UIHintAttribute class defines these properties to be used programmatically: **DefaultProperties**, **ReadOnlyProperties**, **EditProperties**, and **InsertProperties**. Each of these are a collection which take a property name (as a case sensitive string) and a value (in the actual type expected by the property, like an integer or boolean).

Call the method `AddAsDefault()` to add to the **DefaultProperties** collection. Call the method `AddWithMode()` to add to any of the other properties.

[C#]

```
void AddAsDefault(string pPropertyName, object pValue)
void AddWithMode(System.Web.UI.WebControls.DataBoundControlMode pMode,
     string pPropertyName, object pValue)
```

[VB]

```
Sub AddAsDefault(ByVal pPropertyName As String, ByVal pValue As Object)
```

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

```
Sub AddWithMode(ByVal pMode As System.Web.UI.WebControls.DataBoundControlMode, _
     ByVal pPropertyName As String, ByVal pValue As Object)
```

Parameters

pPropertyName

The name of the property on the Field Template. It must be a case sensitive match.

pValue

The value to pass to that property in the actual type of the property.

pMode

Selects one of these properties to hold the property value: **ReadOnlyProperties**, **EditProperties**, and **InsertProperties**.

Example

[C#]

```
using DES=PeterBlum.DES.DynamicData;
…
DES.UIHintAttribute vUIHintAttr = new DES.UIHintAttribute();
vUIHintAttribute.AddAsDefault("DataControlWidth", new Unit(100, UnitType.Pixels));
vUIHintAttribute.AddWithMode("DataControl-BackColor", "aliceblue");
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
…
Dim vUIHintAttr As DES.UIHintAttribute = New DES.UIHintAttribute()
vUIHintAttribute.AddAsDefault("DataControlWidth", New Unit(100, UnitType.Pixels))
vUIHintAttribute.AddWithMode("DataControl-BackColor", "aliceblue")
```

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

DisplayNameAttribute: Friendly column names

By default, ASP.NET Dynamic Data displays the property name from the EntityTable class whereever the column name is needed. Often that name isn't appropriate for your users. Use the DisplayNameAttribute to assign a better name. In fact, it offers two forms of names: the full name and a short name, often used in column titles.

Note: ASP.NET 4 introduces the System.ComponentModel.DataAnnotations.DisplayAttribute which overlaps the DisplayNameAttribute with its DisplayName and ShortDisplayName properties. You can use DisplayAttribute, although it will not support the DES String Lookup System for localization.

There are actually two of these attributes, `System.ComponentModel.DisplayNameAttribute` comes with the native framework and `PeterBlum.DES.DynamicData.DisplayNameAttribute` comes with DES Dynamic Data. DES's attribute is a subclass of the native attribute, adding support for DES's String Lookup System and the short name. DES's attribute is recommended.

The DynamicLabel and DynamicColumnTitle controls will use the values set in the DisplayNameAttribute by setting their **TextSource** property to either `DisplayName` or `ShortDisplayName`.

Note: The DisplayNameAttribute can also be used to rename a table. See "DisplayNameAttribute: Friendly table names".

Using the DisplayNameAttribute

This attribute can be assigned to property names reflecting the column and class names reflecting the table. When assigned to a table, assign to the EntityTable Metadata class.

Here is the syntax for DES's version of this attribute.

[C#]

```
[DES.DisplayName("ColumnName")]
[DES.DisplayName("ColumnName", "LookupID")]
[DES.DisplayName("ColumnName", "LookupID", "ShortName", "ShortLookupID")]
```

[VB]

```
<DES.DisplayName("ColumnName")> _
<DES.DisplayName("ColumnName", "LookupID")> _
<DES.DisplayName("ColumnName", "LookupID", "ShortName", "ShortLookupID")> _
```

The first parameter is the name to show on the web form.

The second parameter is the LookupID into the String Lookup System's Misc string group. If the LookupID cannot be matched into the Misc string group, the first parameter value is used.

The third and forth parameters are the short name and its lookupID respectively.

Properties

The properties on the DES.DisplayNameAttribute are:

- **DisplayName** (string) – The name to show to the user. **MetaColumn.DisplayName** falls back to using **MetaColumn.Name** when this was not supplied by an attribute.

  It is assigned to the **MetaColumn.DisplayName** property.

  Both the DynamicLabel and DynamicColumnTitle controls let you select this in their **TextSource** properties.

- **DisplayNameLookupID** (string) – An alternative to **DisplayName**. When used, the DisplayName is looked up from another source, such as a resource file or database to use as the DisplayName. Uses the String Group StringLookup.StringLookupGroup.Misc.

- **ShortDisplayName** (string) – A shorter version of the name to show to the user. This is often used in column titles.

  ASP.NET 4.0 only It is assigned to the **MetaColumn.ShortDisplayName** property. **MetaColumn.ShortDisplayName** falls back to using **MetaColumn.DisplayName** when this was not supplied by an attribute.

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

Both the DynamicLabel and DynamicColumnTitle controls let you select this in their **TextSource** properties.

- **ShortDisplayNameLookupID** (string) – An alternative to ShortDisplayName. When used, the DisplayName is looked up from another source, such as a resource file or database to use as the DisplayName. Uses the String Group StringLookup.StringLookupGroup.Misc.

Example: Full name

[C#]

```
[DES.DisplayName("Start Date")]
public object StartDate { get; set; }
```

[VB]

```
<DES.DisplayName("Start Date") > _
Public Property StartDate As Object
  Get
    Return Nothing
  End Get
  Set
  End Set
End Property
```

Example: Full name and short name

[C#]

```
[DES.DisplayName("Start Date", "", "Start", "")]
public object StartDate { get; set; }
```

[VB]

```
<DES.DisplayName("Start Date", "", "Start", "") > _
Public Property StartDate As Object
  Get
    Return Nothing
  End Get
  Set
  End Set
End Property
```

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

DescriptionAttribute: A Hint for the user

A good user interface assists the user. One technique is to have a tooltip or hint associated with data entry controls. Use the DescriptionAttribute to assign that description in the metadata. Alternatively, assign it programmatically by using the `CustomizeColumn()` method. See "Using the CustomizeColumn Method".

Note: ASP.NET 4 introduces the System.ComponentModel.DataAnnotations.DisplayAttribute which overlaps the DescriptionAttribute with its Description and Prompt properties. You can use DisplayAttribute, although it will not support the DES String Lookup System for localization.

DES Dynamic Data will setup either a Tooltip or Enhanced Tooltip automatically when a description is available. The Enhanced Tooltip is described in the **Interactive Pages User's Guide**. You can disable this feature by setting the **UseDescriptionAsToolTip** property to `false` on the Customizer control or on any DES Field Template. (See "Setting Properties on the Field Templates".)

There are actually two of these attributes, `System.ComponentModel.DescriptionAttribute` comes with the native framework and `PeterBlum.DES.DynamicData.DescriptionAttribute` comes with DES Dynamic Data. DES's attribute is a subclass of the native attribute, adding support for DES's String Lookup System. Use either one, although if you need strings to be localized, you may prefer the one from DES.

The DynamicLabel and DynamicColumnTitle controls will use the values set in the DescriptionAttribute by setting their **TextSource** property to either `Description` or `Prompt`.

Using the DescriptionAttribute

Here is the syntax for DES's version of this attribute.

[C#]

```
[DES.Description("description text")]
[DES.Description("description text", "LookupID")]
[DES.Description("description text", "LookupID", "prompt text", "PromptLookupID")]
```

[VB]

```
<DES.Description("description text")> _
<DES.Description("description text", "LookupID")> _
<DES.Description("description text", "LookupID", "prompt text", "PromptLookupID")> _
```

The first parameter is the text for the tooltip. The text can contain HTML tags if you intend to display the text in an Enhanced Tooltip.

The second parameter is the LookupID into the String Lookup System's Misc string group. If the LookupID cannot be matched into the Misc string group, the first parameter value is used.

The third and forth parameters are the prompt and its lookupID respectively. The text can contain HTML tags.

Properties

The properties on the DES.DescriptionAttribute are:

- **Description** (string) – The name to show to the user. It is automatically used in tooltips and hints by the Field Templates based on the setting of the **UseDescriptionAsToolTip** property found on each Field Template and on the Customizer control.

  It is assigned to the **MetaColumn.Description** property.

  Both the DynamicLabel and DynamicColumnTitle controls let you select this in their **TextSource** properties.

- **DescriptionLookupID** (string) – An alternative to **Description**. When used, the description is looked up from another source, such as a resource file or database to use as the description. Uses the String Group StringLookup.StringLookupGroup.Misc.

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

- **Prompt** (string) – Another form of description intended for prompts. (Its value is not used for tooltips and hints by the Field Templates.)

  ASP.NET 4.0 only It is assigned to the **MetaColumn.Prompt** property.

  Both the DynamicLabel and DynamicColumnTitle controls let you select this in their **TextSource** properties.

- **PromptLookupID** (string) – An alternative to Prompt. When used, the DisplayName is looked up from another source, such as a resource file or database to use as the DisplayName. Uses the String Group StringLookup.StringLookupGroup.Misc.

Example: Description only

[C#]

```
[DES.Description("The date when the project started.")]
public object StartDate { get; set; }
```

[VB]

```
<DES.Description("The date when the project started.") > _
Public Property StartDate As Object
   Get
      Return Nothing
   End Get
   Set
   End Set
End Property
```

Example: Description and Prompt

[C#]

```
[DES.Description("The date when the project started.", "",
   "Enter the date when the project started.", "")]
public object StartDate { get; set; }
```

[VB]

```
<DES.Description("The date when the project started.", "", _
   "Enter the date when the project started.", "") > _
Public Property StartDate As Object
   Get
      Return Nothing
   End Get
   Set
   End Set
End Property
```

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

Using the CustomizeColumn Method

You can programmatically assign a description by implementing the `PeterBlum.DES.DynamicData.ICustomizeColumn` interface on your EntityTable or the `PeterBlum.DES.DynamicData.IUICustomizeColumn` interface on your web form. See "Customizing Metadata At Runtime". Both define the `CustomizeColumn()` method.

The `CustomizeColumn()` method is passed a `PeterBlum.DES.DynamicData.ModifiableColumn` object. Its **Description** property contains the value from the DescriptionAttribute of the column or "" if there was no DescriptionAttribute.

If you want to set it using the String Lookup System, use the `SetDescriptionByLookupID()` method. See "Method: SetDescriptionByLookupID".

Example

[C#]

```
[MetadataType(typeof(CustomerMetadata))]
public partial class Customer : DES.ICustomizeColumn
{
   public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn, bool pIsNewRow)
   {
      switch (pModifiableColumn.ColumnName)
      {
         case "Region":
            pModifiableColumn.Description = "Enter the country name.";
            break;
      }  // switch
   }
}
```

[VB]

```
<MetadataType(GetType(CustomerMetadata))> _
Partial Public Class Customer _
     Implements DES.ICustomizeColumn

   Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn,_
     ByVal pIsNewRow As Boolean)

      Select Case pModifiableColumn.ColumnName
         Case "Region"
            pModifiableColumn.Description = "Enter the country name."
      End Select
   End Sub
End Class
```

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

Method: SetDescriptionByLookupID

Set the **ModifiableColumn.Description** property using the String Lookup System. The LookupID must be found in the Misc string group. If LookupID does not match any entry in the Misc string group, *pDefault* is used. If *pDefault* is blank, the **Description** property is not changed.

[C#]

```
public void SetDescriptionByLookupID(string pDefault, string pLookupID)
```

[VB]

```
Public Sub SetDescriptionByLookupID(ByVal pDefault As String, _
      ByVal pLookupID As String)
```

Parameters

pDefault

The text to use when the LookupID is not found.

pLookupID

The LookupID used by the String Lookup System to locate an entry in the Misc string group.

Example

[C#]

```
[MetadataType(typeof(CustomerMetadata))]
public partial class Customer : DES.ICustomizeColumn
{
   public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn, bool pIsNewRow)
   {
      switch (pModifiableColumn.ColumnName)
      {
         case "Region":
            pModifiableColumn.SetDescriptionByLookupID("Enter the country name.",
               "CountryDesc");
            break;
      }  // switch
   }
}
```

[VB]

```
<MetadataType(GetType(CustomerMetadata))> _
Partial Public Class Customer _
     Implements DES.ICustomizeColumn

   Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn,_
     ByVal pIsNewRow As Boolean)

      Select Case pModifiableColumn.ColumnName
         Case "Region"
            pModifiableColumn.SetDescriptionByLookupID("Enter the country name.", _
               "CountryDesc")
      End Select
   End Sub
End Class
```

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

DefaultValueAttribute: Initial value when creating a new record

Use the DefaultValueAttribute to define the initial value for the column when in Insert mode (creating a record). Alternatively, assign the default value programmatically by using the `CustomizeColumn()` method. See "Using the CustomizeColumn Method".

The class `System.ComponentModel.DefaultValueAttribute` is documented here: http://msdn.microsoft.com/en-us/library/system.componentmodel.defaultvalueattribute.aspx.

When using the ForeignKey_Edit.ascx Field Template (the column has a one-to-one relationship), use the DefaultValueAttribute to assign a default selection by providing the Value, not the text shown to the user in the dropdownlist.

DES does not offer its own version of this property. Be sure to omit the "DES." text when declaring this attribute.

Example

Suppose the Quantity column accepts an integer. It should have a default value of 1.

[C#]

```
[DefaultValue(1)]
public object Quantity { get; set; }
```

[VB]

```
<DefaultValue(1) > _
Public Property Quantity As Object
   Get
      Return Nothing
   End Get
   Set
   End Set
End Property
```

Example 2: One-To-One relationship default

Suppose the Categories relationship on the Products table should default to the Category "Beverages", whose PrimaryKey ID is "2034". Set the DefaultValue to "2034" (instead of "Beverages").

[C#]

```
[DefaultValue(2034)]
public object Categories { get; set; }
```

[VB]

```
<DefaultValue(2034) > _
Public Property Categories As Object
   Get
      Return Nothing
   End Get
   Set
   End Set
End Property
```

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

Using the CustomizeColumn Method

You can programmatically assign a default value by implementing the `PeterBlum.DES.DynamicData.ICustomizeColumn` interface on your EntityTable or the `PeterBlum.DES.DynamicData.IUICustomizeColumn` interface on your web form. See "Customizing Metadata At Runtime". Both define the `CustomizeColumn()` method.

The `CustomizeColumn()` method is passed a `PeterBlum.DES.DynamicData.ModifiableColumn` object. Its **DefaultValue** property contains the value from the DefaultValueAttribute of the column or `null`/`Nothing` if there was no DefaultValueAttribute. **DefaultValue**'s type is object, so you can assign anything to it.

Example

Suppose the Quantity column accepts an integer. It should have a default value of 1.

[C#]

```
[MetadataType(typeof(ProductMetadata))]
public partial class Product : DES.ICustomizeColumn
{
   public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn, bool pIsNewRow)
   {
      switch (pModifiableColumn.ColumnName)
      {
         case "Quantity":
            pModifiableColumn.DefaultValue = 1;
            break;
      }  // switch
   }
}
```

[VB]

```
<MetadataType(GetType(ProductMetadata))> _
Partial Public Class Product _
     Implements DES.ICustomizeColumn

   Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn,_
     ByVal pIsNewRow As Boolean)

      Select Case pModifiableColumn.ColumnName
         Case "Quantity"
            pModifiableColumn.DefaultValue = 1
      End Select
   End Sub
End Class
```

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

DisplayFormatAttribute: Rules to convert from value to a string

Column values that need to be displayed as strings use the DisplayFormatAttribute to assist in converting from its native type to the string. Alternatively, assign the rules programmatically by using the `CustomizeColumn()` method. See "Using the CustomizeColumn Method".

The idea is the same as with the `String.Format()` method, which takes a pattern and the original value, and returns a string. There are several other rules supplied by the DisplayFormatAttribute. Set them through the parameters of the attribute.

The class `System.ComponentModel.DataAnnotations.DisplayFormatAttribute` is documented here: http://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.displayformatattribute.aspx.

DES does not offer its own version of this property. Be sure to omit the "DES." text when declaring this attribute.

Properties

Note: You must explicitly declare the parameter name in the attribute.

- **DataFormatString** (string) – A formatting string that specifies the display format of items in the column.
- **NullDisplayText** (string) – The caption displayed for a field when the field's value is null.
- **ConvertEmptyStringToNull** (boolean) – Determines whether empty string values ("") are automatically converted to null values when the data field is updated in the data source.
- **ApplyFormatInEditMode** (boolean) – Determines whether the formatting string specified by the **DataFormatString** property is applied to field values when the data-bound control is in edit mode.
- **HtmlEncode** (boolean) – *Requires Dynamic Data v4 or higher*. Indicates if the text should be Html Encoded before inserting into the page. This will replace various symbols with HTML characters so they appear as symbols in the page. It defaults to `true`.

  Note: When using Dynamic Data v3.5, use the CustomizeColumn() method and assign ModifiableColumn.HtmlEncode. Additionally, some Field Templates offer an HtmlEncode property.

For more details, see http://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.displayformatattribute_members.aspx.

Example

Suppose the HireDate column should be formatted as "yyyy-MM-dd".

[C#]

```
[DisplayFormat(DataFormatString="yyyy-MM-dd")]
public object HireDate { get; set; }
```

[VB]

```
<DisplayFormat(DataFormatString:="yyyy-MM-dd") > _
Public Property HireDate As Object
  Get
    Return Nothing
  End Get
  Set
  End Set
End Property
```

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

Using the CustomizeColumn Method

You can programmatically assign these parameters by implementing the `PeterBlum.DES.DynamicData.ICustomizeColumn` interface on your EntityTable or the `PeterBlum.DES.DynamicData.IUICustomizeColumn` interface on your web form. See "Customizing Metadata At Runtime". Both define the `CustomizeColumn()` method.

The `CustomizeColumn()` method is passed a `PeterBlum.DES.DynamicData.ModifiableColumn` object. Use its DataFormatString, NullDisplayText, ConvertEmptyStringToNull, ApplyFormatInEditMode, and HtmlEncode properties.

Example

Support the CustomerName column should be HtmlEncoded (convert HTML tags into displayable text).

[C#]

```
[MetadataType(typeof(CustomerMetadata))]
public partial class Customer : DES.ICustomizeColumn
{
   public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn, bool pIsNewRow)
   {
      switch (pModifiableColumn.ColumnName)
      {
         case "CustomerColumn":
            pModifiableColumn.HtmlEncode = true;
            break;
      }  // switch
   }
}
```

[VB]

```
<MetadataType(GetType(CustomerMetadata))> _
Partial Public Class Customer _
     Implements DES.ICustomizeColumn

   Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn,_
     ByVal pIsNewRow As Boolean)

      Select Case pModifiableColumn.ColumnName
         Case "CustomerColumn"
            pModifiableColumn.HtmlEncode = True
      End Select
   End Sub
End Class
```

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

ScaffoldColumnAttribute: Determining if Scaffolding adds a Column

The ScaffoldColumnAttribute helps Automatic Scaffolding determine if a colum is included or not by setting its **Scaffold** property. See "Automatic Scaffolding Overview".

There are actually two of these attributes, `System.ComponentModel.DataAnnotations.ScaffoldColumnAttribute` comes with the native framework and `PeterBlum.DES.DynamicData.ScaffoldColumnAttribute` ("DES.ScaffoldColumnAttribute") comes with DES Dynamic Data. DES's attribute is a subclass of the native attribute, adding support for the following

- Include or exclude by data entry mode with the **ScaffoldReadOnly**, **ScaffoldEdit**, and **ScaffoldInsert** properties.
- Making a column private so it absolutely cannot be added by Automatic Scaffolding with the **Private** property.
- Changing the column's order with the **Position** and **AfterThisColumn** properties.

Note: The ScaffoldColumnAttribute breaks the separation between business logic and user interface. If you want to maintain the separation, use the ScaffoldSettings property of the DataBoundControlAdapter in your DynamicDataManager or IFieldGeneratorGuidance interface.

ALERT: If you do not use Automatic Scaffolding, this attribute is ignored. In addition, Automatic Scaffolding has a setting to ignore the ScaffoldColumnAttribute. However, that setting respects the Private property.

Using the DES.ScaffoldColumnAttribute

This section documents the DES.ScaffoldColumnAttribute. The class `System.ComponentModel.DataAnnotations.ScaffoldColumnAttribute` is documented here: http://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.scaffoldcolumnattribute.aspx.

Most columns are included by default. (See below.) Generally you specify this attribute when you want to hide a column or to change its position. You can either set its **Scaffold** property to `false`, or use one of these if the data entry mode matters: **ScaffoldReadOnly**, **ScaffoldEdit**, and **ScaffoldInsert**.

Somestimes a property should never be exposed in your user interface, such as a time stamp property. In that case, set the **Private** property to `true`.

The list of properties output by scaffolding follows the order they are defined in your Entity Table class. Often their order is incorrect for display to the user. Use the **AfterThisColumnName** to specify where to reposition this column, based on the name of another column. Use the **Position** property to move its position either relative to its current position (like ahead by 2 or back by 2 which is a value of -2) or a specific position from the start of the list. Set the **AbsolutePositioning** property to false if you want to position relative. Otherwise the **Position** property is from the start of the list.

You can override automatic scaffolding to determine the exact columns you need and their order with the IFieldGeneratorGuidance interface. See "Customizing Automatic Scaffolding". When used, you can tell automatic scaffolding if the ScaffoldColumnAttribute is used to generate an initial list of columns.

How Scaffolding works when there is no ScaffoldColumnAttribute specified

If no ScaffoldColumnAttribute is specified, the following conditions determine the column is being scaffolded by evaluating the MetaColumn's properties in this order.

- Kept: A Field Template name is specified in the UIHintAttribute.
- Omitted: MetaColumn.IsForeignKeyComponent = true (Part of a ForeignKeyColumn)
- Omitted: MetaColumn.IsGenerated = true (The column's data is generated by the database)
- Kept: MetaColumn.IsPrimaryKey = true (The column is a primary key)
- Omitted: MetaColumn.IsCustomProperty = true (The column was created in the EntityTable class, but does not exist in the database)
- Kept: MetaColumnType is one of these: string, char, integer, floating point, bool, DateTime, TimeSpan, or DateTimeOffset.
- Omitted: Anything not listed above.

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

Syntax

Here is the syntax for the native attribute.

The native ScaffoldColumnAttribute

It takes one parameter, a boolean. When `true`, include the column. When `false`, exclude the column.

[C#]

```
[ScaffoldColumn(false)]
```

[VB]

```
<ScaffoldColumn(False)> _
```

The DES.ScaffoldColumnAttribute

[C#]

```
[DES.ScaffoldColumn(false)]   // do not scaffold (can be overridden)
[DES.ScaffoldColumn(ScaffoldEdit=false, ScaffoldInsert=false)]// ReadOnly scaffolds
[DES.ScaffoldColumn(2, true)] // absolute position after the 2nd column
[DES.ScaffoldColumn(-1, false)] // relative position back 1 column
[DES.ScaffoldColumn("Price")] // position after the column named "Price"
[DES.ScaffoldColumn(Private=true)]  // do not scaffold (cannot be overridden)
```

[VB]

```
<DES.ScaffoldColumn(False)> _ ' do not scaffold (can be overridden)
<DES.ScaffoldColumn(ScaffoldEdit:=False, ScaffoldInsert:=False)] ' ReadOnly scaffolds
<DES.ScaffoldColumn(2, True)> _ ' absolute position after the 2nd column
<DES.ScaffoldColumn(-1, False)> _ ' relative position back 1 column
<DES.ScaffoldColumn("Price")> _ ' position after the column named "Price"
<DES.ScaffoldColumn(Private:=True)> _ ' do not scaffold (cannot be overridden)
```

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

Properties of the DES.ScaffoldColumnAttribute

Properties to include or exclude columns

These properties can also be established as parameters on the metadata.

- **Scaffold** (bool) – When `true`, include the column. When `false`, exclude the column. If you assign a value to any of the mode-specific Scaffold properties, that property overrides.

  The ScaffoldColumn attribute includes the column even when other rules would have traditionally excluded it, such as auto-generated with a primarykey (like "CategoryID"). Yet there are some special cases:

  - In DynamicListView, GridView and ListView, string-type columns that allow very long strings are omitted.
  - Children columns are omitted from Insert mode.

- **ScaffoldReadOnly** (bool) – When `true`, include the column in ReadOnly mode. When `false`, exclude the column in ReadOnly Mode. If this property is not explicitly set, it uses the value set in the **Scaffold** property.

- **ScaffoldEdit** (bool) – When `true`, include the column in Edit mode. When `false`, exclude the column in Edit Mode. If this property is not explicitly set, it uses the value set in the **Scaffold** property.

- **ScaffoldInsert** (bool) – When `true`, include the column in Insert mode. When `false`, exclude the column in Insert Mode. If this property is not explicitly set, it uses the value set in the **Scaffold** property.

- **Private** (bool) – The FieldGenerator can be instructed to ignore the **ScaffoldColumnAttribute.Scaffold** property by setting up the IPatternTemplateFieldGeneratorGuidance or IDataControlFieldGeneratorGuidance interface and setting the **Args.UseScaffoldColumnsAttribute** property to `false`. As a result, columns that use the ScaffoldColumnAttribute to hide it will not always hide it. Use the **Private** property to override this behavior. When `true`, this column will NEVER appear in the user interface that uses scaffolding.

  It defaults to `false`.

Properties for positioning

These properties can also be established as parameters on the metadata.

- **Position** (System.Nullable<integer>) – When assigned, position either absolutely or relatively depending on the **AbsolutePositioning** property. When absolute, **Position** is the actual column number, starting at 0. When relative, Position is offset from the initial column position. Use negative values to offset closer to the start.

  It defaults to `null`.

- **AbsolutePositioning** (bool) – When **Position** is assigned, this determines if positioning is absolute or relative. When `true`, it is absolute (where 0 is the first column). When `false`, it is relative and Position can be negative to move the column closer to the first item in the list.

  It defaults to `true`.

- **AfterThisColumnName** (string) – An alternative way to position, by specifying the column name after which this column will appear. The name must be known in the table (although it is case insensitive) or an exception will be thrown at runtime. If you use **Position** and this property, only **Position** will be used.

  It defaults to "".

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

SortableColumnAttribute: Determining sorting rules

The SortableColumnAttribute determines if a column is sortable and any SQL expression is need to sort it. It is used by the DynamicColumnTitle control to setup a user interface for sorting the column. (The user interface provides a clickable column name which toggles sorting and offers a marker identifying the sort direction.)

It has two properties. Use the **Sortable** property to include or exclude sorting. For columns without this attribute, the DynamicColumnTitle will make them sortable if they are numeric, date/time or a short string. Use **Sortable**=`false` to omit those it normally considers sortable.

Use **SortExpression** to describe the sort rules, if they differ from just sorting by the column name.

Since it automatically sorts "short strings", there needs to be a way to determine the maximum length of a short string. By default, it is 100 characters. Change that default using the global property **PeterBlum.DES.DynamicData.SortableColumnAttribute.SortableMaxLength**. Set it in the `Application_Start()` method like this:

```
PeterBlum.DES.DynamicData.SortableColumnAttribute.SortableMaxLength = 20
```

Using the SortableColumnAttribute

Here is the syntax for this attribute.

[C#]

```
[DES.SortableColumn(false)]   // do not sort
[DES.SortableColumn(true)] // sort
[DES.SortableColumn("Price")] // set the SortExpression to the column named "Price"
```

[VB]

```
<DES.SortableColumn(False)> _ ' do not sort
<DES.SortableColumn(True)> _ ' sort
<DES.SortableColumn("Price")> _ ' set the SortExpression to the column named "Price"
```

Properties

These properties can also be established as parameters on the metadata.

- **Sortable** (bool) – When `true`, allow sorting. When `false`, do not sort. It defaults to `true`.
- **SortExpression** (string) – When sorting is supported, this can be a SQL expression used for sorting. If blank, it sorts just the column name. Internally, the DynamicColumnTitle creates a LinkButton with a CommandName of "Sort" and CommandArgument with this value. The DataBound controls recognize this command and invoke their sorting features.

  It will adapt to the correct column name from scaffolding or DataField property if you leave this blank.

  It defaults to "".

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

CategoryAttribute: Assigning Categories to Data Fields

Use the CategoryAttribute to associate a list of category names with a data field. With categories on data fields, you can:

- Scaffold based on certain groups. Perhaps you categorize fields that are always shown with the category named "Always". Use the **FieldGeneratorSettings** property on the DataBoundControlAdapter to specify which categories select data fields in scaffolded.
- Customize the styles of your PatternTemplates by categories in NamedStyles. Suppose you mark a field that reflects the record's name with the category of "Title". Setup a NamedStyle to change any data field with the category of "Title" to show a different font. See "The NamedStyle class".
- Individual Field Templates can customize their output. Suppose that you want to change the style sheet class of a Label control when the CategoryAttribute is defined as "Title". See "Customization based on Categories".

In addition, categories can be used to exclude a data field in the above cases so you select all data fields but the one with a specific category.

The `System.ComponentModel.CategoryAttribute` has one property, **Category**. It takes string where you define the categories. If you want to use multiple categories, define them as a pipe (|) delimited list: "`category1|category2`"

DES does not offer its own version of this property. Be sure to omit the "DES." text when declaring this attribute.

Using the CategoryAttribute

Here is the syntax for this attribute.

[C#]

```
[Category("Title")]
[Category("Title|Always")]
```

[VB]

```
<Category("Title")> _
<Category("Title|Always")> _
```

Properties

These properties can also be established as parameters on the metadata.

- **Category** (string) – The category names. When there are multiple names, use a pipe (|) delimited list. These names will be matched using case insensitive matches.

  It defaults to "".

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

EditableAttribute: Prevent editing on certain columns

Note: This attribute requires ASP.NET 4 or higher.

The System.ComponentModel.DataAnnotations.EditableAttribute offers one way to restrict users from editing specific columns. If you have role-based security, use the ColumnRestrictionAttribute instead of the EditableAttribute.

Dynamic Data uses the EditableAttribute within the FieldTemplateFactory class to retrieve a Field Template using a different mode than what was requested on the DynamicControl's **Mode** property.

DES does not offer its own version of this property. Be sure to omit the "DES." text when declaring this attribute.

Using The EditableAttribute

Properties

Note: You must explicitly declare the parameter name in the attribute.

- **AllowEdit** (Boolean) – Determines if the column allows editing during edit and insert modes. When `true` (the default), editing is permitted.
- **AllowInitialValue** (Boolean) – When **AllowEdit** is `false` and the DynamicControl is in Insert mode, you may prefer to allow the user to enter a value. Effectively, the Field Template is read only during Edit mode and editable in Insert mode. When `true`, editing is allowed in insert mode. It defaults to `false`.

Syntax

Here is the syntax for the EditableAttribute. The one-parameter form is passed the value for the **AllowEdit** property.

[C#]

```
[Editable(false)]
[Editable(AllowEdit=false, AllowInitialValue=true)]
```

[VB]

```
<Editable(False)> _
<Editable(AllowEdit=False, AllowInitialValue=True)> _
```

Example

Suppose the Quantity column can only be entered in insert mode.

[C#]

```
[Editable(AllowEdit=false, AllowInitialValue=true)]
public object Quantity { get; set; }
```

[VB]

```
<Editable(AllowEdit=False, AllowInitialValue=True)> _
Public Property Quantity As Object
  Get
    Return Nothing
  End Get
  Set
  End Set
End Property
```

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

ColumnRestrictionAttribute: Deny access to columns by Role

The ColumnRestrictionAttribute associates security restrictions with columns of a table based on user roles. For an overview of ASP.NET security and roles, see ASP.NET-2.0 Membership Roles and Forms-Authentication, a posting by Scott Guthrie.

This attribute is associated with columns. If you want security on tables, use the TableRestrictionAttribute.

Before these attributes can do any work, you must setup "Setup Security by User Roles". Once setup, automatic scaffolding and the DES.DynamicControl will impose restrictions on the column data. Often that means hiding the data, even the entire column. You can show the column, switching the data into a readonly mode but prevent editing or show a replacement for the data that indicates it has been hidden. See "Showing an alternative interface with a DES.DynamicControl".

Using the DES.ColumnRestrictionAttribute

Add one or more of these attributes, one for each role that needs restrictions. If a role is left unspecified, it usually means that there is no restrictions. However, you can setup Dynamic Data to behave otherwise.

In most sites, roles have priorities, where some users have access to everything, some to nothing, and most to something in between. DES determines your role priorities in the setup for Dynamic Data. As a result, you don't have to worry about the order of the ColumnRestrictionAttributes. However, each ColumnRestrictionAttribute should define a unique Role name.

The restrictions are defined in the `PeterBlum.DES.DynamicData.DenyAccess` enumerated type. It has these values:

- `None` – This does not impose a security restriction. It is often used as a way for methods in this system indicate that there is no security error. Its valid to assign it to the TableRestrictionAttribute and ColumnRestrictionAttribute. This effectively disables the attribute.
- `View` – Cannot view the data in single record or list views. Can still edit, insert, or delete. If you do not want to view, edit, insert or delete, choose `DenyAccess.All`.
- `Edit` – Cannot go into edit mode on an existing record or edit the column. You can still insert. If you want to prevent edit and insert, choose `DenyAccess.Write`.
- `Insert` – Cannot go into insert mode (create a new record)
- `Delete` – Cannot delete a record. Not supported by ColumnRestrictionAttribute.
- `Identify` – Cannot see any presence of this table or column, even in a list of available tables or through a foreign key.
- `Write` – Cannot go into edit or insert modes. Effectively the same as `DenyAccess.Edit OR DenyAccess.Insert` (a bitset OR)
- `AllActions` – Cannot take any of the actions: View, Edit, Insert, or Delete. Can still see the presence of the table or column. (The DenyAcess.Identity restriction is not imposed.) Not supported by ColumnRestrictionAttribute.
- `All` – Cannot take any action with the table or column, including viewing it.

Here is the syntax for this attribute.

[C#]

```
[DES.ColumnRestriction("rolename", DES.DenyAccess.value)]
[DES.ColumnRestriction("Admin", DES.DenyAccess.None)]
[DES.ColumnRestriction("Sales", DES.DenyAccess.Insert | DES.DenyAccess.Edit)]
[DES.ColumnRestriction("Support", DES.DenyAccess.View)]
```

[VB]

```
<DES.ColumnRestriction("rolename", DES.DenyAccess.value),
DES.ColumnRestriction("Admin", DES.DenyAccess.None),
DES.ColumnRestriction("Sales", DES.DenyAccess.Insert Or DES.DenyAccess.Edit),
DES.ColumnRestriction("Support", DES.DenyAccess.View)> _
```

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

Properties

These properties can also be established as parameters on the metadata.

- **Role** (string) – The role name. It must be unique amongst the ColumnRestrictionAttributes on the same table. It will be matched using a case insensitive comparison.
- **Restrictions** (enum PeterBlum.DES.DynamicData.DenyAccess) – Identifies the restrictions for the role. The DenyAccess type supports bitset operations, allowing you to combine several, as shown in the example above where Insert and Edit are combined. While the Delete value may have no meaning to DES Dynamic Data, you can use it for your own purposes.

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

CacheListAttribute: Caching Lists for Enumerated and ForeignKeys

When a Field Template needs to populate a ListBox or DropDownList, it has to gather a list of strings. For Enumerated types, it's a list of strings from the EnumeratedAttribute or EnumeratedStringAttribute. For ForeignKeys, its data from another table. This process happens on each page request when the list control lacks the current data. It's very inefficient when the list of strings rarely changes. The CacheListAttribute lets you cache the list of strings using the ASP.NET Cache class.

Since it uses the `Cache.Insert()` method, it provides properties for many of the `Insert()` method's parameters. The most important is the Key, which is defined by the **CacheKey** property on CacheListAttribute. Each unique list of strings should have a unique **CacheKey**. If several columns use the same list of strings, they can use the same value in the CacheKey property.

When assigned to a column's metadata, the CacheListAttribute has a fixed value for its CacheKey. If the column's list should vary based on the user, you can implement the `CustomizeColumn()` method and modify the CacheKey during the page request. See "Using the CustomizeColumn Method".

Using the CacheListAttribute

Here is the syntax for this attribute.

[C#]

```
[DES.CacheList("cachekey")]
```

```
[DES.CacheList("cachekey", expires_after_seconds,
   System.Web.Caching.CacheItemPriority.Value)]
```

[VB]

```
<DES.CacheList("cachekey") > _
```

```
<DES.CacheList("cachekey", expires_after_seconds, _
   System.Web.Caching.CacheItemPriority.Value)> _
```

Properties

These properties can also be established as parameters on the metadata.

Credit: Some of this text is adapted from Microsoft documentation.

- **CacheKey** (string) – Required for caching. Specifies the Key in the `Cache.Insert()` method.

  Specify unique values for each unique list. If several Columns use the same data for their lists, this can be the same value. It can be customized on each page load using the `CustomizeColumn()` method. See "Using the CustomizeColumn Method".

- **SlidingExpiration** (TimeSpan) – The interval between the time the added object was last accessed and the time at which that object expires. If this value is the equivalent of 20 minutes, the object expires and is removed from the cache 20 minutes after it is last accessed.

  If you are using absolute expiration, the **SlidingExpiration** property must be `Cache.NoSlidingExpiration` which is the default.

  You cannot setup a TimeSpan in the Metadata due to limitations of Attributes. If you want to set this in the metadata, provide a double value containing the number of seconds as the second parameter in the attribute definition as shown here, defining 1 hour (3600 seconds).

  [C#]

  ```
  [DES.CacheList("cachekey", 3600)]
  ```

  [VB]

  ```
  <DES.CacheList("cachekey", 3600) > _
  ```

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

- **AbsoluteExpiration** (DateTime) – The time at which the added object expires and is removed from the cache.

  If you are using sliding expiration, the **AbsoluteExpiration** property must be `Cache.NoAbsoluteExpiration`, which is the default.

  You cannot setup a DateTime in the Metadata due to limitations of Attributes. Instead, you have to assign it programmatically using the `CustomizeColumn()` method. See "Using the CustomizeColumn Method".

- **Priority** (enum System.Web.Caching.CacheItemPriority) – The relative cost of the object, as expressed by the CacheItemPriority enumeration. The cache uses this value when it evicts objects; objects with a lower cost are removed from the cache before objects with a higher cost.

  It defaults to `CacheItemPriority.Normal`. For other values, see http://msdn.microsoft.com/en-us/library/system.web.caching.cacheitempriority.aspx.

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

Using the CustomizeColumn Method

You can programmatically assign or modify the CacheListAttribute by implementing the `PeterBlum.DES.DynamicData.ICustomizeColumn` interface on your EntityTable or the `PeterBlum.DES.DynamicData.IUICustomizeColumn` interface on your web form. See "Customizing Metadata At Runtime". Both define the `CustomizeColumn()` method.

The `CustomizeColumn()` method is passed a `PeterBlum.DES.DynamicData.ModifiableColumn` object. Its property **CacheListAttribute** contains the current CacheListAttribute of the column or `null`/`Nothing` if there was no CacheListAttribute. You can modify the object's properties, including **CacheKey**, or create a new instance and assign it to **ModifiableColumn.CacheListAttribute**.

Example

Set the AbsoluteExpiration to a date one year from today.

[C#]

```
[MetadataType(typeof(CustomerMetadata))]
public partial class Customer : DES.ICustomizeColumn
{
   public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn, bool pIsNewRow)
   {
      switch (pModifiableColumn.ColumnName)
      {
         case "Region":
            pModifiableColumn.CacheListAttribute.AbsoluteExpiration =
               DateTime.Today.AddYear(1);
            break;
      }  // switch
   }
}
public class CustomerMetadata
{
   [DES.EnumeratedString("USA=United States|Canada|Mexico")]
   [DES.CacheList("NorthAmericanCountries", System.Web.Caching.CacheItemPriority.Low)]
   public object Region { get; set; }
}
```

[VB]

```
<MetadataType(GetType(CustomerMetadata))> _
Partial Public Class Customer _
     Implements DES.ICustomizeColumn

   Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn,_
     ByVal pIsNewRow As Boolean)

      Select Case pModifiableColumn.ColumnName
         Case "Region"
            pModifiableColumn.CacheListAttribute.AbsoluteExpiration = _
               DateTime.Today.AddYear(1)
      End Select
   End Sub
End Class
```

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

```
Public Class CustomerMetadata
   <DES.EnumeratedString("USA=United States|Canada|Mexico")> _
   <DES.CacheList("NorthAmericanCountries", System.Web.Caching.CacheItemPriority.Low)>_
   Public Property Region As Object
      Get
         Return Nothing
      End Get
      Set
      End Set
   End Property
End Class
```

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

Customizing Metadata At Runtime

Field Templates rely on the MetaColumn class to determine many of its characteristics, including validators, tooltips, and dependencies. Initially those characteristics are defined using attributes. Attributes create metadata, which is unchangable data until you recompile the application. Often those characteristics must be adjusted at runtime. The `PeterBlum.DES.DynamicData.ICustomizeColumn` and `PeterBlum.DES.DynamicData.IUICustomizeColumn` interfaces let you setup the `CustomizeColumn()` method where you will modify some of the charactertics of a MetaColumn.

- ICustomizeColumn Interface – Use on the EntityTable class. Generally used to modify business logic, such as validation attributes and dependencies. See "Implementing the ICustomizeColumn Interface".
- IUICustomizeColumn Interface – Use in the code of User Controls, Pages, and Master Pages. Generallyused to modify interface rules, such the Description used as a tooltip. See "Implementing the IUICustomizeColumn Interface".

It is important to realize that the MetaColumn's properties can only be established as the application starts up. That's because the MetaColumn is global data. If you edit a property, such as MetaColumn.Description, it will affect every thread using that application. To work around this, DES provides the `PeterBlum.DES.DynamicData.ModifiableColumn` class. See "Understanding the ModifiableColumn class".

Note: DES Dynamic Data supports edits made to the MetaColumn during Application_Start() using the InMemoryMetadataManager.AddColumnAttributes() method found in Dynamic Data Futures (7/16/08 build).

Click on any of these topics to jump to them:

- Understanding the ModifiableColumn class
- Implementing the ICustomizeColumn Interface
- Implementing the IUICustomizeColumn Interface

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

Understanding the ModifiableColumn class

The `PeterBlum.DES.DynamicData.ModifiableColumn` class is passed into the `CustomizeColumn()` method. You can modify most of its properties, although you should treat its **StaticColumn** property as readonly because that is the actual global MetaColumn object.

You will notice a lack of several properties from MetaColumns such as **DisplayName** and **UIHint**. The `CustomizeColumn()` method is called to prepare an initialized Field Template, which is too late for establishing these values. For example, UIHint determines which Field Template to load.

ModifiableColumn properties that you should not modify

These properties will assist your `CustomizeColumn()` method to determine what actions to take. Generally you will evaluate **TableName**, **ColumnName** and **RowMode**.

- **ColumnName** (string) – A readonly property that you use to determine which column is being processed in `CustomizeColumn()`.
- **TableName** (string) – A readonly property that you use to determine which table is being processed in `CustomizeColumn()`. You will use this with the IUICustomizeColumn interface which accepts all tables whereas the ICustomizeColumn interface is connected to a specific table and doesn't need this property.
- **StaticColumn** (MetaColumn) – A reference to the actual MetaColumn. Since this object is global data, do not modify it!
- **RowMode** (enum System.Web.UI.WebControls.DataBoundControlMode) – The mode of the row from the DataBound control: `ReadOnly`, `Edit`, or `Insert`. On list/grid style controls, each row can have a different mode.

  Use this property to optimize your `CustomizeColumn()` method. Normally you will not prepare for validation when the mode is ReadOnly. Its likely that other attributes only need to be established based on the mode as well.
- **RowData** (object) – The instance of the EntityTable class that is associated with this row. Use it to prepare attributes based on data specific rules. (When using the ICustomizeColumn interface, the method is already exposing the same object with its `this` (C#) or `Me` (VB) keyword.)
- **DataControlAdapter** (PeterBlum.DES.DynamicData.BaseDataControlAdapter) – The DataControlAdapter class that is managing the DataBound control. You should not modify its properties.

ModifiableColumn properties that you can modify

- **IsRequired** (boolean) – When `true`, the column requires a value. Exposes MetaColumn.IsRequired.
- **IsReadOnly** (boolean) – When `true`, the column is readonly. DES's Field Templates will apply a different user interface to these columns when in edit mode. Exposes MetaColumn.IsReadOnly.
- **MaxLength** (integer) – The maximum length of a string-type column. When 0, there is no maximum. *Warning: it is possible to assign a length that exceeds the database's defined length limit.* Exposes MetaColumn.MaxLength.
- **Description** (string) – The text used for tooltips and hints. It initially is defined by a DescriptionAttribute. See "DescriptionAttribute: A Hint for the user". If you want to set it using the String Lookup System, see "Method: SetDescriptionByLookupID". Exposes MetaColumn.Description.

  For more, see "Using the CustomizeColumn Method" on the DescriptionAttribute.
- **DefaultValue** (string) – The default value for the column when in Insert mode (creating a record). It is initially defined by the DefaultValueAttribute. See "DefaultValueAttribute: Initial value when creating a new record". Exposes MetaColumn.DefaultValue.

  For more, see "Using the CustomizeColumn Method" on the DefaultValueAttribute.

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

- **ValidationAttributes** (PeterBlum.DES.DynamicData.ValidationAttribute) – A collection of ValidationAttributes that OVERRIDE the ValidationAttributes previously determined from metadata. This list is initially empty. Instead of editing it directly, please use these methods:
  - `GetExistingValidationAttribute()` – Returns the ValidationAttribute from the type requested or `null` if none defined. See "Method: GetExistingValidationAttribute".
  - `GetValidationAttribute()` – Returns the ValidationAttribute from the type requested. If it was not declared previously, it will be created and added to **ValidationAttributes**. See "Method: GetValidationAttribute".
  - `RemoveValidationAttribute()` – Removes a ValidationAttribute declared previously. See "Method: RemoveValidationAttribute"

  For more, see "Using the CustomizeColumn Method" on the ValidationAttribute.

- **DataTypeAttribute** (PeterBlum.DES.DynamicData.DataTypeAttribute) – In addition to clarifying the data type, the DES.DataTypeAttribute can tell the Field Template how to process its data. For example, if currency values allow negatives. See "DataTypeAttributes: The DataTypes used by your Business logic".

  Use the **DataTypeAttribute** property to modify the existing DataTypeAttribute's properties. You can also create a new instance and assign it to the DataTypeAttribute property. Just be aware that the Field Template has already been created and this new DataTypeAttribute cannot change the Field Template itself.

  When assigning properties, be sure to typecast to the actual DataTypeAttribute class so you can access its properties.

  If no DataTypeAttribute was assigned to the column's metadata, **DataTypeAttribute** will return `null`/`Nothing`.

- **DependencyAttribute** (PeterBlum.DES.DynamicData.BaseDependencyAttribute) – The current DependencyAttribute class assigned to this column or null. You can edit it or replace it. These methods are available to make editing it easier:
  - `CreateRequiredDependencyAttribute()` – Creates a new RequiredDependencyAttribute for use with this column. See "Method: CreateRequiredDependencyAttribute".
  - `CreateConditionDependencyAttribute()` – Creates a new ConditionDependencyAttribute for use with this column. See "Method: CreateConditionDependencyAttribute".
  - `RemoveDependencyAttribute()` – Eliminates a DependencyAttribute already on this column. See "Method: RemoveDependencyAttribute".

- **UIHintAttribute** (PeterBlum.DES.DynamicData.UIHintAttribute) – In addition to specifying a Field Template, the DES.UIHintAttribute can supply values to the Field Template's properties. For example, all Field Templates offer the property DataControlWidth. See "Setting properties on Field Templates".

  Use the **UIHintAttribute** property to modify the existing UIHintAttribute's DefaultProperties, ReadOnlyProperties, EditProperties, and InsertProperties properties. You can also create a new instance and assign it to the **UIHintAttribute** property. Just be aware that the Field Template has already been created, so edits to **UIHint**, **ReadOnlyUIHint**, **EditUIHint**, and **InsertUIHint** properties cannot change the Field Template itself.

  If no UIHintAttribute was assigned to the column's metadata, **UIHintAttribute** will return `null`/`Nothing`.

- **CacheListAttribute** (PeterBlum.DES.DynamicData.CacheListAttribute) – The CacheListAttribute for this column. Typically you modify the **CacheKey** property on this attribute to allow different versions of the cached data based on the situation.

- **ApplyFormatInEditMode** (Boolean) – Determines whether the formatting string specified by the **DataFormatString** property is applied when the DynamicControl/Field is in Edit mode. The same property exists on DynamicField.

- **DataFormatString** (string) – Determines the string that specifies the display format for the value of the field. This is most useful on date, time and numeric columns displayed within a Label (usually ReadOnly mode). When using any of DES's TextBoxes for dates, times or numbers, they have built in formatting rules that may conflict with a setting here.

- **NullDisplayText** (string) – Determines the text displayed for a field when the field's value is `null`.

- **ConvertEmptyStringToNull** (Boolean) – Determines whether empty strings are converted to a null value when saving. The same property exists on DynamicField.

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

- **HtmlEncode** (Boolean) – Determines whether field values are HTML-encoded before they are displayed. This is most useful when the field is ReadOnly and shown in a Label. When false, HTML tags are used as real HTML to supply formatting. When true, they are show as text on the page.

  Suggested usage: when a string column incorporates HTML, you need to determine if the user reviews the HTML tags or sees the formatted information.

  Note: Several Field Template classes have their own HtmlEncode property that can override this property when their value is not set to TrueFalseDefault.Default.

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

Implementing the ICustomizeColumn Interface

The `PeterBlum.DES.DynamicData.ICustomizeColumn` Interface should be assigned to the EntityTable Class (actually the partial class you have created in the code file containing the metadata). Once assigned, create the `CustomizeColumn()` method. The method has this format.

[C#]

```
public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn)
```

[VB]

```
Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn)
```

Parameters

pModifiableColumn

You will modify elements of this object. See "Understanding the ModifiableColumn class". See the example below.

Use its **ColumnName** property to determine which column you have been passed.

Use the **RowMode** property to avoid modifying attributes on a mode that doesn't need those attributes, such as `ReadOnly` mode doesn't need validators.

When implementing this method for the IUICustomizeColumn interface, use the **TableName** property to determine the table associated with the column pass.

See also on individual Attributes:

- ValidatorAttributes
- DataTypeAttributes
- UIHintAttribute
- EnumeratedAttribute and EnumeratedStringAttribute
- DependencyAttributes
- DescriptionAttribute
- DefaultValueAttribute
- CacheListAttribute

ALERT: If your EntityTable Class does not have a parameterless constructor, or you want to handle initialization differently than with the parameterless constructor, see "Creating Instances of the EntityTable classes".

SEE THE EXAMPLE ON THE NEXT PAGE

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

Example

[C#]

```
public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn)
{
   switch (pModifiableColumn.ColumnName)
   {
      case "PostalCode":
         pModifiableColumn.DefaultValue = "00000";
         pModifiableColumn.MaxLength = 9;
         break;
      case "City": // add validator except in ReadOnly mode
         if (pModifiableColumn.RowMode == DataBoundControlMode.ReadOnly)
            break;
         DES.CharacterSetAttribute vCSA =
            new DES.CharacterSetAttribute(true, true, false, true, false, "-.'");
         vCSA.DiacriticLetters = true;
         pModifiableColumn.ValidationAttributes.Add(vCSA);
         break;
      case "Fax": // dependency – require the Phone number column
         pModifiableColumn.DependencyAttribute =
            new DES.ConditionDependencyAttribute(
               new PeterBlum.DES.RequiredTextCondition("Phone"));
         break;
   }  // switch
}
```

[VB]

```
Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn) _
  Implements DES.ICustomizeColumn.CustomizeColumn

  Select Case pModifiableColumn.ColumnName
     Case "PostalCode"
        pModifiableColumn.DefaultValue = "00000"
        pModifiableColumn.MaxLength = 9
     Case "City"    ' add validator except in ReadOnly mode
        If pModifiableColumn.RowMode <> DataBoundControlMode.ReadOnly Then
         Dim vCSA As DES.CharacterSetAttribute = _
            New DES.CharacterSetAttribute(True, True, False, True, False, "-.'")
         vCSA.DiacriticLetters = True
         pModifiableColumn.ValidationAttributes.Add(vCSA)
        End If
     Case "Fax"    ' dependency – require the Phone number column
        pModifiableColumn.DependencyAttribute =
         New DES.ConditionDependencyAttribute(
            New PeterBlum.DES.RequiredTextCondition("Phone"))
    End Select
End Sub
```

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

Implementing the IUICustomizeColumn Interface

The `PeterBlum.DES.DynamicData.IUICustomizeColumn` Interface can be assigned to almost any object, although typically you put it on a UserControl, Page, or MasterPage class. It is based on the ICustomizeColumn Interface (in the previous topic) with the addition of another method, `GetUICustomizeColumnBehavior()`.

The `CustomizeColumn()` method is setup similarly too, but it is called for every column on every table. So it needs to test **ModifiableColumn.TableName** and **ModifiableColumn.ColumnName**.

Click on any of these topics to jump to them:

- Where to implement IUICustomizeColumn
- Adding the IUICustomizeColumn Interface
- The GetUICustomizeColumnBehavior method
- The CustomizeColumn method

Where to implement IUICustomizeColumn

As an interface, you attach it to a class then create the list of methods within that class. You already have some classes created where you should consider attaching it: the Page, the MasterPage, or any UserControl containing the DynamicDataManager control. If you use any of these, DES will find and use it. You can also explicitly assign the class to the Customizer control's **UICustomizeColumnOwner** property.

How DES locates your interface

You may have this interface declared in a few classes, such as both the MasterPage and UserControl containing a DynamicDataManager. If you let DES search for the interface, it always searches in this order:

1. User control with DynamicDataManager
2. Master Page
3. Page

How to specify which object to use

If you need to specify which of these to use, assign the object to the Customizer control's **UICustomizeColumnOwner** property.

Always set the **UICustomizeColumnOwner** property in the DynamicDataManager's Init event handler method or `Page_Init()` method.

[C#]

```
Customizer1.UICustomizeColumnOwner = this;
```

[VB]

```
Customizer1.UICustomizeColumnOwner = Me
```

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

Adding the IUICustomizeColumn Interface

1. Start by using the `PeterBlum.DES.DynamicData` namespace. It's easiest to map it to the name "DES" just like was done in the code file containing the EntityTable Metadata Classes.

   [C#]

   ```
   using DES = PeterBlum.DES.DynamicData;
   ```

   [VB]

   ```
   Import DES = PeterBlum.DES.DynamicData
   ```

2. Assign the IUICustomizeColumn interface to the desired class.

   [C#]

   ```
   public class WebForm1 : System.Web.UI.WebControls.Page,
       DES.IUICustomizeColumn
   ```

   [VB]

   ```
   Public Class WebForm1
      Implements System.Web.UI.WebControls.Page, _
      Implements DES.IUICustomizeColumn
   ```

3. Implement the `GetUICustomizeColumnBehavior()` method. See "The GetUICustomizeColumnBehavior method".

4. Implement the `CustomizeColumn()` method. See "The CustomizeColumn method".

5. If its not already on the page, add the Customizer control. It is the conduit between your method and the code that invokes it. When using Page Templates, typically the Customizer is added to the MasterPage.

6. DES Dynamic Data will discover your control or page class that implements IUICustomizeColumn so long as that class is a User Control that contains the DES DynamicDataManager control, the MasterPage, or the Page. See "How DES locates your interface".

   If it cannot be discovered, assign the object to the Customizer's **UICustomizeColumnOwner** property in either the DynamicDataManager's Init event handler or the `Page_Init()` method like this:

   [C#]

   ```
   Customizer1.UICustomizeColumnOwner = this;
   ```

   [VB]

   ```
   Customizer1.UICustomizeColumnOwner = Me
   ```

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

The GetUICustomizeColumnBehavior method

This is a method required by the IUICustomizeColumn interface.

There can be two `CustomizeColumn()` methods evaluating a ModifiableColumn object.

- EntityTable implementing ICustomizeColumn
- Pages and UserControls implementing IUICustomizeColumn

When both are setup, you need to determine the order they are called because one can overwrite the edits made by the other. Use the `GetUICustomizeColumnBehavior()` method to determine the order and if the ICustomizeColumn interface is just ignored. Here is its definition.

[C#]

```
PeterBlum.DES.DynamicData.UICustomizeColumnBehavior GetUICustomizeColumnBehavior();
```

[VB]

```
Function GetUICustomizeColumnBehavior() _
  As PeterBlum.DES.DynamicData.UICustomizeColumnBehavior
```

It returns the enumerated type `PeterBlum.DES.DynamicData.UICustomizeColumnBehavior`. Return one of these values:

- `IgnoreTable` – Only call `IUICustomizeColumn.CustomizeColumn()`. Ignore the EntityTable.
- `BeforeTable` – Call `IUICustomizeColumn.CustomizeColumn()` before `ICustomizeColumn.CustomizeColumn()`.
- `AfterTable` – Call `IUICustomizeColumn.CustomizeColumn()` after `ICustomizeColumn.CustomizeColumn()`. Allows overriding the EntityTable settings. This is the recommended setting.

Example

[C#]

```
public DES.UICustomizeColumnBehavior GetUICustomizeColumnBehavior()
{
  return DES.UICustomizeColumnBehavior.AfterTable;
}
```

[VB]

```
Public Function GetUICustomizeColumnBehavior() As DES.UICustomizeColumnBehavior _
     Implements DES.IUICustomizeColumn.GetUICustomizeColumnBehavior

  Return DES.UICustomizeColumnBehavior.AfterTable
End Function
```

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

The CustomizeColumn method

This is a method required by the IUICustomizeColumn interface.

The CustomizeColumn method has this format.

[C#]

```
public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn,
  bool pIsNewRow)
```

[VB]

```
Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn, _
  ByVal pIsNewRow As Boolean)
```

Parameters

pModifiableColumn

You will modify elements of this object. Use its **TableName** and **ColumnName** properties to determine which column you have been passed. See the example below.

pIsNewRow

When `true`, the EntityTable class's data will be used in Insert mode. When `false`, it is not in Insert mode.

See also on individual Attributes:

- ValidatorAttributes
- DataTypeAttributes
- UIHintAttribute
- EnumeratedAttribute and EnumeratedStringAttribute
- DependencyAttributes
- DescriptionAttribute
- DefaultValueAttribute
- CacheListAttribute

SEE THE EXAMPLE ON THE NEXT PAGE

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

Example

Special handling of several columns in the Customer table.

[C#]

```
public void CustomizeColumn(DES.ModifiableColumn pModifiableColumn)
{
   switch (pModifiableColumn.TableName)
   {
      case "Customer":
         switch (pModifiableColumn.ColumnName)
         {
            case "PostalCode":
               pModifiableColumn.DefaultValue = "00000";
               pModifiableColumn.MaxLength = 9;
               break;
            case "City": // add validator except in ReadOnly mode
               if (pModifiableColumn.RowMode == DataBoundControlMode.ReadOnly)
                  break;
               DES.CharacterSetAttribute vCSA =
                  new DES.CharacterSetAttribute(true, true, false,
                     true, false, "-.'");
               vCSA.DiacriticLetters = true;
               pModifiableColumn.ValidationAttributes.Add(vCSA);
               break;
            case "Fax": // dependency – require the Phone number column
               pModifiableColumn.DependencyAttribute =
                  new DES.ConditionDependencyAttribute(
                     new PeterBlum.DES.RequiredTextCondition("Phone"));
               break;
         }  // switch ColumnName
         break;
   }  // switch TableName
}
```

[VB]

```
Public Sub CustomizeColumn(ByVal pModifiableColumn As DES.ModifiableColumn) _
  Implements DES.ICustomizeColumn.CustomizeColumn

   Select Case pModifiableColumn.TableName
      Case "Customer"
        Select Case pModifiableColumn.ColumnName
           Case "PostalCode"
              pModifiableColumn.DefaultValue = "00000"
              pModifiableColumn.MaxLength = 9
           Case "City"    ' add validator except in ReadOnly mode
             If pModifiableColumn.RowMode <> DataBoundControlMode.ReadOnly Then
               Dim vCSA As DES.CharacterSetAttribute = _
                 New DES.CharacterSetAttribute(True, True, False, True, False, "-.'")
               vCSA.DiacriticLetters = True
               pModifiableColumn.ValidationAttributes.Add(vCSA)
             End If
           Case "Fax"    ' dependency – require the Phone number column
              pModifiableColumn.DependencyAttribute =
               New DES.ConditionDependencyAttribute(
                 New PeterBlum.DES.RequiredTextCondition("Phone"))
         End Select
   End Select  ' TableName
End Sub
```

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

Applying Attributes To EntityTables

In the previous section, you applied attributes to individual columns. There are business logic attributes available for the class definition of each EntityTable too. They also help Dynamic Data as it assists in building the user interface.

Remember that there are now 3 classes representing each table:

- The original "EntityTable" that was generated for you by LINQ to SQL or ADO.NET Entity Framework
- The Partial class with the same name, in the code file you created
- The EntityTable Metadata Class, in the code file you created.

You will be applying attributes to the EntityTable Metadata Class only. Start by defining properties on that class, as described in the next topic.

Click on any of these topics to jump to them:

- Steps for defining your business logic
- DisplayNameAttribute: Friendly table names
- DisplayColumnAttribute: Selecting a column name for foreign keys
- ScaffoldTableAttribute: Determining if a table appears in the UI
- TableRestrictionAttribute: Deny access to tables by Role

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

DisplayNameAttribute: Friendly table names

By default, ASP.NET Dynamic Data displays the property name from the EntityTable class whereever the table name is needed. Often that name isn't appropriate for your users. Use the DisplayNameAttribute to assign a better name.

There are actually two of these attributes, `System.ComponentModel.DisplayNameAttribute` comes with the native framework and `PeterBlum.DES.DynamicData.DisplayNameAttribute` comes with DES Dynamic Data. DES's attribute is a subclass of the native attribute, adding support for DES's String Lookup System. Use either one, although if you need strings to be localized, you may prefer the one from DES.

Note: The DisplayNameAttribute can also be used to rename a column. See "DisplayNameAttribute: Friendly column names".

Note: Dynamic Data Futures (7/16/08 build) provides another DisplayNameAttribute, LocalizedDisplayNameAttribute. It is supported by DES Dynamic Data, but uses a different mechanism for string lookups.

Using the DisplayNameAttribute

This attribute can be assigned to the class names reflecting the table. Assign it to the EntityTable Metadata class.

Here is the syntax for DES's version of this attribute.

[C#]

```
[DES.DisplayName("TableName")]
[DES.DisplayName("TableName", "LookupID")]
```

[VB]

```
<DES.DisplayName("TableName")> _
<DES.DisplayName("TableName", "LookupID")> _
```

The first parameter is the name to show on the web form.

The second parameter is the LookupID into the String Lookup System's Misc string group. If the LookupID cannot be matched into the Misc string group, the first parameter value is used.

Example

These should be assigned to the EntityTable MetaData class.

[C#]

```
[MetadataType(typeof(CategoryMetadata))]
public partial class Category
{
}

[DES.DisplayName("Categories")]
public class CategoryMetadata
{

}
```

[VB]

```
<MetadataType(GetType(CategoryMetadata))> _
Partial Public Class Category
End Class

<DisplayName("Categories")> _
Public Class CategoryMetadata
End Class
```

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

DisplayColumnAttribute: Selecting a column name for foreign keys

Specifies which column to display (in filters or in foreign key links). By default, Dynamic Data uses the first column of type string that it finds. It also lets you change the default sorting, to specify a different column and sort order. This attribute is specified on the EntityTable class definition, not on the properties of the class.

The class `System.ComponentModel.DataAnnotations.DisplayColumnAttribute` is documented here: http://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.displaycolumnattribute.aspx.

DES does not offer its own version of this property. Be sure to omit the "DES." text when declaring this attribute.

Specifying an alternative column name

ForeignKeys and filters refer to another table. The first string-type column from that table may not contain the right data. For example, you have a table with these columns: FirstName, LastName, City, State, and PostalCode. If you don't specify the DisplayColumnAttribute, the ForeignKey and filter will list data from the FirstName column. Here's how to display the LastName column:

[C#]

```
[DisplayColumn("LastName")]
partial class Customer { }
```

[VB]

```
<DisplayColumn("LastName")> _
Partial Class Customer
End Class
```

Customizing the data shown

Suppose you wanted to show both the first and last name in this format: "LastName, FirstName".

Create a new string-type property on the EntityTable class declared in your metadata code file. It will gather the values from other columns and format it as you prefer. Then specify that property name in the DisplayColumn attribute.

[C#]

```
partial public string CombinedNames
{
  get { return this.LastName + ", " + this.FirstName; }
}
```

[VB]

```
Partial Public ReadOnly Property CombinedNames As String
  Get
    Return Me.LastName + ", " + Me.FirstName
  End Get
End Property
```

Sorting by a different column

The DisplayColumnAttribute takes up to 3 parameters. The first is the alternative column for data. The second and third relate to sorting.

[C#]

```
[DisplayColumn("Display Column Name", "Sort Column Name", true|false)]
partial class Customer { }
```

[VB]

```
<DisplayColumn("Display Column Name", "Sort Column Name", True|False)> _
Partial Class Customer
End Class
```

The second parameter is the column name to sort upon. If omitted, it sorts by the first string-type column in the table.

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

The third parameter is a boolean. When `true`, it sorts in descending order. When `false` or omitted, it sorts in ascending order.

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

ScaffoldTableAttribute: Determining if a table appears in the UI

The ScaffoldTableAttribute lets you selectively remove tables from the automatic scaffolding feature. It takes one parameter, a boolean. When `true`, include the table (the default). When `false`, exclude the table.

Generally you assign this attribute to the EntityTable class (the partial class in the code file with the metadata). However, it also works when assigned to the EntityTable Metadata Class.

The class `System.ComponentModel.DataAnnotations.ScaffoldTableAttribute` is documented here: http://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.scaffoldtableattribute.aspx.

DES does not offer its own version of this property. Be sure to omit the "DES." text when declaring this attribute.

Using the ScaffoldTableAttribute

Here is the syntax for this attribute.

[C#]

```
[ScaffoldTable(false)]
public partial class Products {}
```

[VB]

```
<ScaffoldColumn(False)> _
Partial Public Class Products
End Products
```

Properties

These properties can also be established as parameters on the metadata.

- **Scaffold** (bool) –When `false`, exclude the table.

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

TableRestrictionAttribute: Deny access to tables by Role

The TableRestrictionAttribute associates security restrictions with a table based on user roles. For an overview of ASP.NET security and roles, see ASP.NET-2.0 Membership Roles and Forms-Authentication, a posting by Scott Guthrie.

This attribute is associated with tables. If you want security on individual columns, use the ColumnRestrictionAttribute.

Generally you assign this attribute to the EntityTable class (the partial class in the code file with the metadata). However, it also works when assigned to the EntityTable Metadata Class.

Before these attributes can do any work, you must setup "Setup Security by User Roles". Once setup, routing actions will not allow access to pages based on the table and action; buttons for actions may be hidden; and pages can prevent users from seeing secure data.

Using the DES.TableRestrictionAttribute

Add one or more of these attributes, one for each role that needs restrictions. If a role is left unspecified, it usually means that there is no restrictions. However, you can setup Dynamic Data to behave otherwise.

In most sites, roles have priorities, where some users have access to everything, some to nothing, and most to something in between. DES determines your role priorities in the setup for Dynamic Data. As a result, you don't have to worry about the order of the TableRestrictionAttributes. However, each TableRestrictionAttribute should define a unique Role name.

The restrictions are defined in the `PeterBlum.DES.DynamicData.DenyAccess` enumerated type. It has these values:

- `None` – This does not impose a security restriction. It is often used as a way for methods in this system indicate that there is no security error. Its valid to assign it to the TableRestrictionAttribute and ColumnRestrictionAttribute. This effectively disables the attribute.
- `View` – Cannot view the data in single record or list views. Can still edit, insert, or delete. If you do not want to view, edit, insert or delete, choose `DenyAccess.All`.
- `Edit` – Cannot go into edit mode on an existing record or edit the column. You can still insert. If you want to prevent edit and insert, choose `DenyAccess.Write`.
- `Insert` – Cannot go into insert mode (create a new record)
- `Delete` – Cannot delete a record. Not supported by ColumnRestrictionAttribute.
- `Identify` – Cannot see any presence of this table or column, even in a list of available tables or through a foreign key.
- `Write` – Cannot go into edit or insert modes. Effectively the same as `DenyAccess.Edit OR DenyAccess.Insert` (a bitset OR)
- `AllActions` – Cannot take any of the actions: View, Edit, Insert, or Delete. Can still see the presence of the table or column. (The DenyAcess.Identity restriction is not imposed.)
- `All` – Cannot take any action with the table or column, including viewing it.

Here is the syntax for this attribute.

[C#]

```
[DES.TableRestriction("rolename", DES.DenyAccess.value)]
[DES.TableRestriction("Admin", DES.DenyAccess.None)]
[DES.TableRestriction("Sales", DES.DenyAccess.Insert | DES.DenyAccess.Delete)]
[DES.TableRestriction("Support", DES.DenyAccess.Write)]
public partial class Products {}
```

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

[VB]

```
<DES.TableRestriction("rolename", DES.DenyAccess.value),
DES.TableRestriction("Admin", DES.DenyAccess.None),
DES.TableRestriction("Sales", DES.DenyAccess.Insert Or DES.DenyAccess.Delete),
DES.TableRestriction("Support", DES.DenyAccess.Write)> _
Partial Public Class Products
End Products
```

Properties

These properties can also be established as parameters on the metadata.

- **Role** (string) – The role name. It must be unique amongst the TableRestrictionAttributes on the same table. It will be matched using a case insensitive comparison.
- **Restrictions** (enum PeterBlum.DES.DynamicData.DenyAccess) – Identifies the restrictions for the role. The DenyAccess type supports bitset operations, allowing you to combine several, as shown in the example above where Insert and Delete are combined.

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

Defining Validators at Runtime: ICreateValidatorWebControls Interface

Instead of using a ValidationAttribute to create the Validator web control for you, your EntityTable class can implement the ICreateValidatorWebControls interface to create actual DES Validator web controls. This is recommended when there is no ValidationAttribute type for the desired Validator.

The `PeterBlum.DES.DynamicData.ICreateValidatorWebControls` Interface must be assigned to the partial class for the desired EntityTable found in the same code file as your EntityTable Metadata Classes.

Adding the ICreateValidatorWebControls Interface

1. Start by using the `PeterBlum.DES.DynamicData` namespace. It's easiest to map it to the name "DES" just like was done in the code file containing the EntityTable Metadata Classes.

   [C#]

   ```
   using DES = PeterBlum.DES.DynamicData;
   ```

   [VB]

   ```
   Import DES = PeterBlum.DES.DynamicData
   ```

2. Assign the ICreateValidatorWebControls interface to the desired EntityTable partial class. *This is NOT the class that contains metadata on its properties.*

   [C#]

   ```
   public partial class Product : DES.ICreateValidatorWebControls
   ```

   [VB]

   ```
   Partial Public Class Product _
      Implements DES.ICreateValidatorWebControls
   ```

3. Implement the `CreateValidatorWebControls()` method. See "The CreateValidatorWebControls method".

4. If your EntityTable Class does not have a parameterless constructor, or you want to handle initialization differently than with the parameterless constructor, see "Creating Instances of the EntityTable classes".

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

The CreateValidatorWebControls method

This is a method required by the ICreateValidatorWebControls interface.

The `CreateValidatorWebControls()` method is passed a List that accepts DES Validators (any subclass of `PeterBlum.DES.BaseAnyValidator`). Your task is to create Validator objects that are setup as follows:

- Assign properties for conditions, such as the **Minimum** and **Maximum** properties on a RangeValidator
- Assign the Column Name value to the **ControlIDToEvaluate** and **SecondControlIDToEvaluate** properties. DES Dynamic Data will convert those names into the actual control ID for you.
- Optionally assign **ErrorMessage** and **SummaryErrorMessage** properties. If left blank, values will be retrieved from the Customizer control, if it has any for the given type, or use the defaults for the ValidationAttribute type that works with the validator you are creating. If none of these are available, the ErrorMessage is set to "ERROR".

You do not need to take any of these actions because they are done automatically for you:

- Assign the control's ID.
- Setup the ErrorFormatter.
- Setup the Validation Group.
- Call any method on the Field Template to attach the validator, such as `SetUpDESValidator()`.
- Add the Validator control to the Page's control tree.

Once created, add it to the method's *pList* parameter.

Here is the function format.

[C#]

```
public void CreateValidatorWebControls(
   List<PeterBlum.DES.BaseAnyValidator> pValidators,
   MetaColumn pColumn, DES.DESFieldTemplateUserControl pFieldTemplate)
```

[VB]

```
Public Sub CreateValidatorWebControls(
   ByVal pValidators As List(Of PeterBlum.DES.BaseAnyValidator),
   ByVal pColumn As MetaColumn,
   ByVal pFieldTemplate As DES.DESFieldTemplateUserControl)
```

Parameters

pList

Add instances of DES Validator controls. See above for details on what properties to set and what to ignore.

pColumn

The MetaColumn that is being prepared. Use pColumn.Name property to determine whether your code needs to take any action.

pFieldTemplate

The Field Template that will get this validator.

ALERT: If your EntityTable Class does not have a parameterless constructor, or you want to handle initialization differently than with the parameterless constructor, see "Creating Instances of the EntityTable classes".

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

Example

Add the RangeValidator to the ReorderLevel column.

[C#]

```
public void CreateValidatorWebControls(
   List<PeterBlum.DES.BaseAnyValidator> pValidators,
   MetaColumn pColumn, DES.DESFieldTemplateUserControl pFTUC)
{
   if (pColumn.Name == "ReorderLevel")
   {
      PeterBlum.DES.RangeValidator vRangeValidator =
         new PeterBlum.DES.RangeValidator();
      vRangeValidator.ControlIDToEvaluate = "ReorderLevel"; // Column Name
      vRangeValidator.MinimumAsNative = 1;
      vRangeValidator.MaximumAsNative = 10;
      vRangeValidator.DataType = "Integer";
      vRangeValidator.ErrorMessage = "Reorder Level must be between 1 and 10";
      pValidators.Add(vRangeValidator);
   }
}
```

[VB]

```
Public Sub CreateValidatorWebControls(
   ByVal pValidators As List(Of PeterBlum.DES.BaseAnyValidator),
   ByVal pColumn As MetaColumn,
   ByVal pFieldTemplate As DES.DESFieldTemplateUserControl) _

      Implements DES.ICreateValidatorWebControls

   If pColumn.Name = "ReorderLevel" Then
      Dim vRangeValidator As PeterBlum.DES.RangeValidator = _
         New PeterBlum.DES.RangeValidator()
      vRangeValidator.ControlIDToEvaluate = "ReorderLevel" ' Column Name
      vRangeValidator.MinimumAsNative = 1
      vRangeValidator.MaximumAsNative = 10
      vRangeValidator.DataType = "Integer"
      vRangeValidator.ErrorMessage = "Reorder Level must be between 1 and 10"
      pValidators.Add(vRangeValidator)
   End If
End Sub
```

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

Creating Calculated and Custom Columns

You can define additional properties on the EntityTable class (specifically the partial class in the code file containing the EntityTable metadata). While they cannot interact with your database, they can get data for display using Field Templates. Your properties only need the ScaffoldColumnAttribute or UIHintAttribute to tell the automatic scaffolding it belongs in your user interface. Other attributes are supported too.

You can setup a readonly property by defining only its GET selector. Its value will still appear in Edit and Insert modes, but using a ReadOnly interface.

If you define a SET selector, its code should update another property that is connected to the database in order to be written out.

Note: When using Versatile DataSources, your EntityTable can define calculated columns whose data is created by the database server. These properties must have a GET and SET selector.

Example

Calculate the person's age given today's date and the birthdate. The EntityTable has a database driven column called BirthDate.

[C#]

```
[ScaffoldColumn(true)]
public int Age
{
   get
   {
      TimeSpan vAgeInDays = DateTime.Today.Subtract(this.BirthDate);
      return Convert.ToInt32(Math.Floor(vAgeInDays.Days / 364.25));
   }
}
```

[VB]

```
<ScaffoldColumn(True)> _
Public Property Age As Integer
   Get

      Dim vAgeInDays As TimeSpan = DateTime.Today.Subtract(Me.BirthDate)
      Return Convert.ToInt32(Math.Floor(vAgeInDays.Days / 364.25))
   End Get
End Property
```

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

Creating Instances of the EntityTable classes

Normally instances of your EntityTable classes are created by the DataSource control. However, DES Dynamic Data creates an instance of these objects in the following case:

- On a postback to prepare for validation. At this time, the DataSource has not been queried for the objects. Validation is going to look at the data posted back.
- You have implemented the ICustomizeColumn interface or ICreateValidatorWebControls interface on the EntityTable class. These are both setup in preparation for postback.

This is a special case indeed and in many cases, DES Dynamic Data still does its work automatically. It calls a parameterless constructor on your EntityTable class and the object is ready to use. However, you may need to assist it in these situations:

- Your EntityTable class does not have a public parameterless constructor.
- You want to call a different constructor (even if it has a parameterless constructor).
- The EntityTable instance needs additional initialization that isn't handled by the constructor.

The IEntityTypeCreator Interface

Use the `PeterBlum.DES.DynamicData.IEntityTypeCreator` Interface to deliver an instance of the EntityTable object. Its `CreateEntityType()` method lets you call any constructor desired. Because it is on your UserControl, MasterPage, or Page object, it has access to information about the current page request. It is also passed stateful information about the MetaTable, DataBound Control, DataSource, and DataBoundControlAdapter, so you can gather the necessary information to setup your EntityTable object.

Where to implement IEntityTypeCreator

As an interface, you attach it to a class then create the list of methods within that class. You already have some classes created where you should consider attaching it: the Page, the MasterPage, or any UserControl containing the DynamicDataManager control. If you use any of these, DES will find and use it. You can also explicitly assign the class to the Customizer control's **EntityTypeCreatorOwner** property.

How DES locates your interface

You may have this interface declared in a few classes, such as both the MasterPage and UserControl containing a DynamicDataManager. If you let DES search for the interface, it always searches in this order:

1. User control with DynamicDataManager
2. Master Page
3. Page

How to specify which object to use

If you need to specify which of these to use, assign the object to the Customizer control's **EntityTypeCreatorOwner** property.

Always set the **EntityTypeCreatorOwner** property in the DynamicDataManager's Init event handler method or `Page_Init()` method.

[C#]

```
Customizer1.EntityTypeCreatorOwner = this;
```

[VB]

```
Customizer1.EntityTypeCreatorOwner = Me
```

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

Adding the IEntityTypeCreator Interface

1. Start by using the `PeterBlum.DES.DynamicData` namespace. It's easiest to map it to the name "DES" just like was done in the code file containing the EntityTable Metadata Classes.

   [C#]

   ```
   using DES = PeterBlum.DES.DynamicData;
   ```

   [VB]

   ```
   Import DES = PeterBlum.DES.DynamicData
   ```

2. Assign the IEntityTypeCreator interface to the desired class.

   [C#]

   ```
   public class WebForm1 : System.Web.UI.WebControls.Page,
       DES.IEntityTypeCreator
   ```

   [VB]

   ```
   Public Class WebForm1
      Implements System.Web.UI.WebControls.Page, _
      Implements DES.IEntityTypeCreator
   ```

3. Implement the `CreateEntityType()` method. See "The CreateEntityType method".

4. If its not already on the page, add the Customizer control. It is the conduit between your method and the code that invokes it. When using Page Templates, typically the Customizer is added to the MasterPage.

5. DES Dynamic Data will discover your control or page class that implements IEntityTypeCreator so long as that class is a User Control that contains the DES DynamicDataManager control, the MasterPage, or the Page. See "How DES locates your interface".

   If it cannot be discovered, assign the object to the Customizer's **EntityTypeCreatorOwner** property in either the DynamicDataManager's Init event handler or the `Page_Init()` method like this:

   [C#]

   ```
   Customizer1.IEntityTypeCreator = this;
   ```

   [VB]

   ```
   Customizer1.IEntityTypeCreator = Me
   ```

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

The CreateEntityType method

This is a method required by the IEntityTypeCreator interface.

Create an instance of *pArgs*.EntityType or whatever instance you prefer given the arguments. This will be used for validation after postback.

[C#]

```
object CreateEntityType(PeterBlum.DES.DynamicData.EntityTypeCreatorArgs pArgs);
```

[VB]

```
Function CreateEntityType( _
   ByVal pArgs As PeterBlum.DES.DynamicData.EntityTypeCreatorArgs) _
   As Object
```

Parameters

pArgs

Identifies stateful information used to Identifies the MetaTable, TableName, EntityType, DataBoundControlAdapter, DataBoundControl, and DataSource to assist you in creating your EntityType object.

Returns

An instance of the EntityType identified by *pArgs*.EntityType or `null`/`Nothing` if the caller can create it using the *pArgs*.EntityType type and a constructor with no parameters.

Example

[C#]

```
public object CreateEntityType(PeterBlum.DES.DynamicData.EntityTypeCreatorArgs pArgs)
{
   if (pArgs.EntityType == typeof(Product))
   {
      Product vProduct = new Product(true);
      vProduct.UnitPrice = 10.0;
      return vProduct;
   }
   return null;
}
```

[VB]

```
Public Function CreateEntityType (_
   ByVal pArgs As PeterBlum.DES.DynamicData.EntityTypeCreatorArgs) As Object
      Implements DES.IEntityTypeCreator.CreateEntityType

   If pArgs.EntityType = TypeOf(Product) Then

      Dim vProduct As Product = New Product(True)
      vProduct.UnitPrice = 10.0
      Return vProduct
   End If
   Return Nothing;
End Function
```

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

Setup Security by User Roles

When your web application imposes restrictions by roles, you must do some additional setup with DES Dynamic Data before it follows your restrictions.

- Create a RestrictionManager object and associate it with your MetaModel object in `Application_Start()`. See "Global setup for Security Roles".
- Assign TableRestrictionAttributes and ColumnRestrictionAttributes to your EntityTable metadata to associate roles with their restrictions on tables and columns. See "TableRestrictionAttribute: Deny access to tables by Role" and "ColumnRestrictionAttribute: Deny access to columns by Role".
- Modify the user interface of pages that use Dynamic Data features to hide features based on the restrictions. (You don't have to handle hiding buttons built into the DataBound controls or DynamicControlPanel.) See "Customizing the User Interface to Respect Security".

Click on any of these topics to jump to them:

- Features
- Global setup for Security Roles
  - Setup when using FormsAuthentication
  - Setup using an alternative RestrictionManager
  - Blocking restricted URLs
- Customizing the User Interface to Respect Security
  - Securing pages with DataBound controls
  - Hide tables listed in the Entry Point Page
  - Showing an alternative interface with a DES.DynamicControl
  - Customizing Automatic Scaffolding
- Using the RestrictionManager in your own code

Features

DES Dynamic Data's security model is a very rich tool with these features:

- Supports ASP.NET FormAuthentication and custom security systems.
- Roles can be prioritized so that users under multiple roles can be given preferential access if their higher priority role allows access and lower priority role does not.
- Roles are associated with a variety of restrictions to give granular control over access: view, edit, insert, delete, etc.
- When a restriction is imposed, the data can be hidden, but that isn't the only approach. If edit mode is blocked, the data can optionally appear in read-only mode. If a field is blocked, it can be replaced by an alternative interface, such as text saying "blocked" with another style sheet.
- When a restriction is imposed, the associated command buttons are automatically hidden. (Requires the use of DynamicButtons controls.)
- Automatic scaffolding eliminates columns that are restricted.
- Url Routing can be setup to return a 404 error if the user requests a restricted URL.

In addition to security restrictions, roles can be used to change the styles of the user interface within NamedStyles.

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

Global setup for Security Roles

When security restrictions are imposed, you must create a RestrictionManager object and attach it to the MetaModel object during `Application_Start()`. The RestrictionManager provides these services:

- Determines if security restrictions are active.
- Determines if a desired role name is currently in use.
- Determines if a restriction is imposed for a table or column for the current logged in user.

DES Dynamic Data interacts with the RestrictionManager automatically, modifying the user interface as needed for many cases. For other cases, your code can use these services. See "Using the RestrictionManager in your own code".

When using ASP.NET FormsAuthentication, use the class `PeterBlum.DES.DynamicData.FormsAuthenticationRestrictionManager` as your RestrictionManager. If you have another system for role and login management, create a subclass of `PeterBlum.DES.DynamicData.BaseRestrictionManager`. See "Setup using an alternative RestrictionManager".

Setup when using FormsAuthentication

Before taking these steps, be sure you have established the `PeterBlum.DES.DynamicData.DESMetaModel` in `Application_Start()` or `RegisterRoutes()` methods. See "Create the DESMetaModel object".

In the same method as where you created DESMetaModel, create a `PeterBlum.DES.DynamicData.FormsAuthenticationRestrictionManager` object and assign it to the DESMetaModel's **RestrictionManager** property after the call to `MetaModel.RegisterContext()`.

[C#]

```
using DES=PeterBlum.DES.DynamicData;
...
DES.MetaModel vMetaModel = new DES.DESMetaModel();

vMetaModel.RegisterContext(typeof(DataContext),
   new ContextConfiguration() { ScaffoldAllTables = true });

// this must be assigned after RegisterContext.
vMetaModel.RestrictionManager = new DES.FormsAuthenticationRestrictionManager(
     PrioritizedRoles, RoleSearchMode, WhenRoleIsMissing);
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
...
Dim vMetaModel As DES.MetaModel = New DES.DESMetaModel()

vMetaModel.RegisterContext(GetType(DataContext),  _
   New ContextConfiguration() With { .ScaffoldAllTables = True })

' this must be assigned after RegisterContext.
vMetaModel.RestrictionManager = New DES.FormsAuthenticationRestrictionManager( _
     PrioritizedRoles, RoleSearchMode, WhenRoleIsMissing)
```

CONTINUED ON THE NEXT PAGE

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

Here are the parameters for the constructor:

- **PrioritizedRoles** (string) – A pipe delimited list of all roles used by your system. It imposes a priority on roles by assigning the first role the highest priority and the last the lowest priority. This is used when a user has two or more roles, to determine which roles impose restrictions.
- **RoleSearchMode** (enum PeterBlum.DES.DynamicData.RoleSearchMode) – Determines how to search through the user's roles and TableRestrictionAttributes or ColumnRestrictionAttributes in the metadata to determine the source of the restriction.

  The `PeterBlum.DES.DynamicData.RolePriorityMode` enumerated type has these values:

  - `Highest` - Only the highest priority Role is evaluated.
  - `Lowest` - Only the lowest priority Role is evaluated.
  - `FirstFound` - Search through all of the user's roles in high to low priority order until one is defined as a TableRestrictionAttribute or ColumnRestrictionAttribute in the metadata.
  - `LastFound` - Search through all of the user's roles in low to high priority order until one is defined as a TableRestrictionAttribute or ColumnRestrictionAttribute in the metadata.
  - `CombineStrict` - Search through all of the user's roles. Collect all restrictions from all defined TableRestrictionAttributes or ColumnRestrictionAttributes in the metadata.
  - `CombineLoose` - Search through all of the user's roles. Collect all non-restrictions from all defined TableRestrictionAttributes or ColumnRestrictionAttributes in the metadata.

  It defaults to `RoleSearchMode.FirstFound`.

  Each of these searches may fail to find a match. Use the **WhenRoleIsMissing** property to handle that case.

  If none of these RoleSearchModes fits your scenario, you can subclass FormsAuthenticationRestrictionManager and override its `GetRestrictionsByRole()` method. See “Subclassing to define restrictions by role”.

- **WhenRoleIsMissing** (enum System.Web.Configuration.AuthorizationRuleAction) – After searching through roles based on the **RoleSearchMode**, if no Role was found in the TableRestrictionAttributes or ColumnRestrictionAttributes, this determines if the restriction allows or denies access.

  When `AuthorizationRuleAction.Allow`, allow access. Do not restrict.

  When `AuthorizationRuleAction.Deny`, deny access. Impose a restriction

  It defaults to `AuthorizationRuleAction.Allow`.

EXAMPLES ARE ON THE NEXT PAGE

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

Examples

Here is the data used to form the table "Restrictions by mode" below.

The table "Products" has these restrictions by role:		Users have these roles:	
Admin:	*not specified*	John:	Admin
Finance:	Insert	Steve:	Finance, Admin
Purchasing:	*not specified*	Mary:	Purchasing
Training:	Insert, Delete	Tom:	Sales, Arts
Sales:	Edit, Insert, Delete	Larry:	Purchasing, Finance, Arts
Arts:	View, Edit, Insert, Delete	Linda:	Training, Arts
		Don:	Purchasing, Training, Arts

Restrictions by mode – WhenRoleIsMissing=Allow

```
vMetaModel.RestrictionManager = new DES.FormsAuthenticationRestrictionManager(
     "Admin|Finance|Purchasing|Sales|Training|Arts", DES.RoleSearchMode.value,
     System.Web.Configuration.AuthorizationRuleAction.value)
```

	Highest	Lowest	FirstFound	LastFound	CombineStrict	CombineLoose
John	None	None	None|All*	None|All*	None|All*	None|All*
Steve	None	Insert	Insert	Insert	Insert	Insert
Mary	None	None	None|All*	None|All*	None|All*	None|All*
Tom	Edit, Insert, Delete	View, Edit, Insert, Delete	Edit, Insert, Delete	View, Edit, Insert, Delete	View, Edit, Insert, Delete	Edit, Insert, Delete
Larry	None	View, Edit, Insert, Delete	Insert	View, Edit, Insert, Delete	View, Edit, Insert, Delete	Insert
Linda	Insert, Delete	View, Edit, Insert, Delete	Insert, Delete	View, Edit, Insert, Delete	View, Edit, Insert, Delete	Insert, Delete
Don	None	View, Edit, Insert, Delete	Insert, Delete	View, Edit, Insert, Delete	View, Edit, Insert, Delete	Insert, Delete

The colors indicate the source for the restriction value.

* Those in black did not match to a role. They are determined by **WhenRoleIsMissing** where None is used for WhenRoleIsMissing=Allow.

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

Subclassing to define restrictions by role

When the FormsAuthenticationRestrictionManager's rules for determining which restrictions apply do no work for you, subclass it and override its `GetRestrictionsByRole()` method. Then assign that subclass to the **DESMetaModel.RestrictionManager** property in `Application_Start()` or `RegisterRoutes()` method. This section discusses the `GetRestrictionsByRole()` method.

The method is passed a list of roles for the currently logged in user, ordered by the prioritized roles that were defined in the constructor. It is also passed a list of attributes, either TableRestrictionAttributes or ColumnRestrictionAttributes, that are found on the table or column element.

Your task is to return a bitset describing the restrictions. The DenyAccess type accepts a bitset of flags, such as DenyAccess.View OR DenyAccess.Insert.

[C#]

```
protected virtual PeterBlum.DES.DynamicData.DenyAccess GetRestrictionsByRole(
   List<string> pPrioritizedRolesByUser,
   PeterBlum.DES.DynamicData.RestrictionAttributes pAttributes,
   object pMetaSource)
```

[VB]

```
Protected Overridable Function GetRestrictionsByRole( _
   ByVal pPrioritizedRolesByUser As List Of String, _
   ByVal pAttributes As PeterBlum.DES.DynamicData.RestrictionAttributes, _
   ByVal pMetaSource As Object) As PeterBlum.DES.DynamicData.DenyAccess
```

Parameters

pPrioritizedRolesByUser

A list of the roles for the user ordered by the prioritized roles that were defined in the constructor. If the user has no roles, expect it to be passed an empty list.

pAttributes

A list of either TableRestrictionAttributes or ColumnRestrictionAttributes, that are found on the table or column element. Their order is undefined. The RestrictionAttributes type offers the `FindByRole("role name")` method which returns an attribute that matches to the role name (case insensitive) or `null`/`Nothing` if no matching role was found.

pMetaSource

The MetaColumn or MetaTable object associated with this request. Check its type to determine which.

Returns

A bitset of restrictions based on the `PeterBlum.DES.DynamicData.DenyAccess` type. Return `DenyAccess.None` if there are no restrictions. Return `DenyAccess.All` if the user is fully restrictioned. Build bitsets using the bitwise or operator for other cases:

[C#]

```
DenyAccess.View | DenyAccess.Edit | DenyAccess.Insert
```

[VB]

```
DenyAccess.View Or DenyAccess.Edit Or DenyAccess.Insert
```

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

Example

This example forces the Admin role to always return DenyNone, and use the ancestor to follow the RoleSearchMode and WhenRoleIsMissing properties.

[C#]

```
protected override DenyAccess GetRestrictionsByRole(
   List<string> pPrioritizedRolesByUser,
   RestrictionAttributes pAttributes, object pMetaSource)
{
   if (pPrioritizedRolesByUser.Contains("Admin"))
      return DenyAccess.None;
   return base.GetRestrictionsByRole(pPrioritizedRolesByUser,
      pAttributes, pMetaSource);
}
```

[VB]

```
Protected Overrides Function GetRestrictionsByRole( _
   ByVal pPrioritizedRolesByUser As List Of String, _
   ByVal pAttributes As PeterBlum.DES.DynamicData.RestrictionAttributes, _
   ByVal pMetaSource As Object) As PeterBlum.DES.DynamicData.DenyAccess

   If pPrioritizedRolesByUser.Contains("Admin") Then
      Return DenyAccess.None
   Else
      Return MyBase.GetRestrictionsByRole(pPrioritizedRolesByUser,
         pAttributes, pMetaSource)
   End If
End Function
```

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

Setup using an alternative RestrictionManager

When you are not using ASP.NET FormsAuthentication, you can supply your own class for the **RestrictionManager** property on the DESMetaModel. Subclass from `PeterBlum.DES.DynamicData.BaseRestrictionManager` and implement the following methods, which are all required.

```
/// <summary>
/// Called to allow any initialization needed.
/// </summary>
/// <param name="pMetaModel"></param>
      abstract public void Initialize(MetaModel pMetaModel);

/// <summary>
/// Indicates if restriction features are enabled. When using the System.Web.Security.Roles,
/// this is the value of Roles.Enabled.
/// </summary>
      abstract public bool Enabled { get; }

/// <summary>
/// Returns the DenyAccess flags assocated with the current user.
/// </summary>
/// <remarks>
/// <para>The result is a bitset of flags to indicate all available actions.</para>
/// </remarks>
/// <param name="pMetaTable">The table being evaluated.</param>
/// <returns>If DenyAccess.None, then there are no restrictions.</returns>
      abstract public DenyAccess IdentifyTableRestrictions(MetaTable pMetaTable);

/// <summary>
/// Returns the DenyAccess flags assocated with the current user.
/// </summary>
/// <remarks>
/// <para>The result is a bitset of flags to indicate all available actions.</para>
/// </remarks>
/// <param name="pMetaColumn">The column being evaluated.</param>
/// <returns>If DenyAccess.None, then there are no restrictions.</returns>
      abstract public DenyAccess IdentifyColumnRestrictions(MetaColumn pMetaColumn);

/// <summary>
/// Tests the current user's roles against a list of one or more roles.
/// </summary>
/// <param name="pRoles">If there are multiple roles, use a pipe delimited list.
/// Role matching is case insensitive.</param>
/// <returns>When true, there is a match to at least one role passed in.
/// When false, there are no matched roles.</returns>
      abstract public bool HasRole(string pRoles);
```

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

Blocking restricted URLs

Suppose you restrict access to Edit mode on the Products table. You will want a URL like http://www.mydomain.com/Products/Edit.aspx?ProductID=1 to reject the user in some way. One way to reject is to let the Url Routing feature evaluate the URL and return a 404 error to the browser if the URL is restricted. This section describes how to set it up.

Yet this is *not* the recommended approach. A 404 error is not a very good user interface. A better approach is to add security code into the destination page (like Edit.aspx) that will take a more useful action, like throwing an exception with a helpful error message or hiding the DataBound control but otherwise showing itself.

Before taking these steps, be sure you have established the `PeterBlum.DES.DynamicData.DESMetaModel` in `Application_Start()` or `RegisterRoutes()` methods. See "Setup when using FormsAuthentication".

The task is to add the `PeterBlum.DES.DynamicData.RestrictionDynamicDataRouteHandler` class to the **RouteHandler** property of any DynamicDataRoute object declared in `Application_Start()`.

In the same method as where you created DESMetaModel, create a `PeterBlum.DES.DynamicData.FormsAuthenticationRestrictionManager` object and assign it to the DESMetaModel's **RestrictionManager** property after the call to `MetaModel.RegisterContext()`.

[C#]

```
using DES=PeterBlum.DES.DynamicData;
...
DES.MetaModel vMetaModel = new DES.DESMetaModel();
vMetaModel.RegisterContext(typeof(DataContext),
   new ContextConfiguration() { ScaffoldAllTables = true });
vMetaModel.RestrictionManager = new DES.FormsAuthenticationRestrictionManager(
      "Admin|Finance|Purchasing|Sales|Training|Arts", DES.RoleSearchMode.Highest,
      System.Web.Configuration.AuthorizationRuleAction.Allow);
routes.Add(new DynamicDataRoute("{table}/{action}.aspx") {
      Constraints = new RouteValueDictionary(
         new { action = "List|Details|Edit|Insert" }),
      RouteHandler= new DES.RestrictionDynamicDataRouteHandler(),
      Model = vMetaModel
});
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
...
Dim vMetaModel As DES.MetaModel = New DES.DESMetaModel()
vMetaModel.RegisterContext(GetType(DataContext),  _
   New ContextConfiguration() With { .ScaffoldAllTables = True })
vMetaModel.RestrictionManager = New DES.FormsAuthenticationRestrictionManager( _
      "Admin|Finance|Purchasing|Sales|Training|Arts", DES.RoleSearchMode.Highest, _
      System.Web.Configuration.AuthorizationRuleAction.Allow)
routes.Add(New DynamicDataRoute("{table}/{action}.aspx") With { _
   .Constraints = New RouteValueDictionary( _
      New With {.Action = "List|Details|Edit|Insert"}), _
   .RouteHandler= New DES.RestrictionDynamicDataRouteHandler(),  _
   .Model = vMetaModel
})
```

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

Customizing the User Interface to Respect Security

Here are the typical user interface elements that need to respect security. Some are handled automatically. Others require some setup here:

- Urls that specify a restricted path. You can either hook up the RestrictionDynamicDataHandler to the DynamicDataRoute to get a 404 error (see "Blocking restricted URLs"), or can introduce security on individual pages. See "Securing pages with DataBound controls".
- Buttons are hidden. This happens automatically when using the DynamicButtons control and the buttons of the DynamicControlPanel. Unless you are using GridView and DetailsView controls, you probably are using these types of buttons already.
- The Entry Point Page (Default.aspx) lists tables. If you want to hide tables with the DenyAccess.Identify restriction, see "Hide tables listed in the Entry Point Page".
- The DES.DynamicControls will automatically hide columns that are inappropriate for the given data entry mode. You may wish to override this, allowing the columns to be included, but replacing them with an alternative interface. See "Showing an alternative interface with a DES.DynamicControl".
- Automatic scaffolding will respect the ColumnRestrictionAttribute, automatically omitting columns that are inappropriate for the given data entry mode. You may wish to override this, allowing the columns to be included, but replacing them with an alternative interface. See "Customizing Automatic Scaffolding".

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

Securing pages with DataBound controls

When a page is loaded, it should confirm that its contents is appropriate for the given user. In the case of Dynamic Data, that means hiding the data within the DataBound control when the table it displays is restricted.

Detecting a restriction is easy. In `Page_Load()` or `Page_Init()`, just call the static method `DESMetaModel.CanTakeActionOnTable()` with the MetaTable and a DenyAccess setting. If that function returns false, hide the data. Here are a few approaches to consider:

- Throw an exception
- Redirect to another page
- Replace the DataBound control with alternative HTML
- Change the DataSource's query

Throw an exception

Let the exception deliver an error message to the user. The System.UnauthorizedAccessException class is suitable for this type of exception.

[C#]

```
using DES=PeterBlum.DES.DynamicData;
...
MetaTable vMetaTable = YourDataSource.GetTable();
if (!DES.DESMetaModel.CanTakeActionOnTable(vMetaTable, DES.DenyAccess.View))
   throw new UnauthorizedAccessException("You cannot view this page.");
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
...
Dim vMetaTable As MetaTable = YourDataSource.GetTable()
If Not DES.DESMetaModel.CanTakeActionOnTable(vMetaTable, DES.DenyAccess.View) Then
   Throw New UnauthorizedAccessException("You cannot view this page.")
End If
```

Redirect to another page

Use `Response.Redirect()` to show a page with suitable information.

[C#]

```
using DES=PeterBlum.DES.DynamicData;
...
MetaTable vMetaTable = YourDataSource.GetTable();
if (!DES.DESMetaModel.CanTakeActionOnTable(vMetaTable, DES.DenyAccess.View))
   Response.Redirect("~/Errors/RestrictedPage.aspx");
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
...
Dim vMetaTable As MetaTable = YourDataSource.GetTable()
If Not DES.DESMetaModel.CanTakeActionOnTable(vMetaTable, DES.DenyAccess.View) Then
   Response.Redirect("~/Errors/RestrictedPage.aspx")
End If
```

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

Replace the DataBound control with alternative HTML

In addition to your DataBound control, add a Panel control with the content to be displayed. Set **Visible** to `false` on the Panel.

Your code should change the visibility of both the DataBound control and Panel, to hide one and show the other.

```
<desDD:DynamicFormView ID="RecordDetails" runat="server"
  DataSourceID="RecordDetailsDataSource" DefaultMode="Edit"
  AutoGenerateButtons="Last" >
</desDD:DynamicFormView>
<asp:Panel ID="ErrorPanel" runat="server" Visible="false">
   You cannot view this data.
</asp:Panel>
```

[C#]

```
using DES=PeterBlum.DES.DynamicData;
...
MetaTable vMetaTable = YourDataSource.GetTable();
if (!DES.DESMetaModel.CanTakeActionOnTable(vMetaTable, DES.DenyAccess.View))
{
   RecordDetails.Visible = false;
   ErrorPanel.Visible = true;
}
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
...
Dim vMetaTable As MetaTable = YourDataSource.GetTable()
If Not DES.DESMetaModel.CanTakeActionOnTable(vMetaTable, DES.DenyAccess.View) Then
   RecordDetails.Visible = False
   ErrorPanel.Visible = True
End If
```

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

Change the DataSource's query

The DataBound control usually has an interface for when there no records shown. Change the DataSource's query to prevent any records from being shown.

```
<asp:LinqDataSource ID="YourDataSource" runat="server" EnableUpdate="true">
    <WhereParameters>
        <asp:DynamicQueryStringParameter />
    </WhereParameters>
</asp:LinqDataSource>
```

[C#]

```
using DES=PeterBlum.DES.DynamicData;
...
MetaTable vMetaTable = YourDataSource.GetTable();
if (!DES.DESMetaModel.CanTakeActionOnTable(vMetaTable, DES.DenyAccess.Edit))
{
  RecordDetailsDataSource.WhereParameters.Clear();
  RecordDetailsDataSource.Where = "ProductID < 0";
}
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
...
Dim vMetaTable As MetaTable = YourDataSource.GetTable()
If Not DES.DESMetaModel.CanTakeActionOnTable(vMetaTable, DES.DenyAccess.Edit) Then
  RecordDetailsDataSource.WhereParameters.Clear()
  RecordDetailsDataSource.Where = "ProductID < 0"
End If
```

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

Hide tables listed in the Entry Point Page

The Entry Point Page that comes with Dynamic Data is **Default.aspx**. It lists the available tables by retrieving a list from the MetaModel.**VisibleTables** property. The DESMetaModel class provides a new property **VisibleTablesForProperty**. Switch to using that in the code behind.

[C#]

```
System.Collections.IList visibleTables = null;
PeterBlum.DES.DynamicData.DESMetaModel vDESMetaModel =
  MetaModel.Default as PeterBlum.DES.DynamicData.DESMetaModel;
if (vDESMetaModel != null)
  visibleTables = vDESMetaModel.VisibleTablesForRole;
else
  visibleTables = MetaModel.Default.VisibleTables;
if (visibleTables.Count == 0) {
  throw new InvalidOperationException("There are no accessible tables.
    Make sure that at least one data model is registered in Global.asax and
    scaffolding is enabled or implement custom pages.");
}
Menu1.DataSource = visibleTables;
Menu1.DataBind();
```

[VB]

```
Dim visibleTables As System.Collections.IList = Nothing
Dim vDESMetaModel As PeterBlum.DES.DynamicData.DESMetaModel = _
  TryCast(MetaModel.[Default], PeterBlum.DES.DynamicData.DESMetaModel)
If vDESMetaModel IsNot Nothing Then
  visibleTables = vDESMetaModel.VisibleTablesForRole
Else
  visibleTables = MetaModel.[Default].VisibleTables
End If
If visibleTables.Count = 0 Then
  Throw New InvalidOperationException("There are no accessible tables.
      Make sure that at least one data model is registered in Global.asax and
      scaffolding is enabled or implement custom pages.")
End If
Menu1.DataSource = visibleTables
Menu1.DataBind()
```

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

Showing an alternative interface with a DES.DynamicControl

The native DynamicControl will show the contents of whatever column you supply in the mode specified in its **Mode** property. It has no concern for your security restrictions. The DES.DynamicControl handles security much better. By default, it will not output a Field Template when it detects a restriction on the column (through a ColumnRestrictionAttribute). You may appreciate a different interface. It offers these:

- When in Edit mode, show the data in read only mode instead. See "Edit mode reverts to ReadOnly mode".
- When in ReadOnly mode, display a User Control whose contents are displayed instead of the data. It may say something like "hidden" and change the style sheet for the container of the field to a grey background. See "ReadOnly mode shows a User Control"

When the column is displaying data of another table, it will check that table's TableRestrictionAttribute to see if it has `DenyAccess.Identify` rights. When it does, the DES.DynamicControl will hide the Field Template. When in ReadOnly mode, it can display the same User Control described above.

Edit mode reverts to ReadOnly mode

1. Add the Customizer control to the page if its not already there. See "Using the Customizer Control".
2. Set the property **DenyEditRestrictionShowsReadOnly** to `true` on the Customizer control.
3. You can select whether the non-editable version of the data is shown as a Label or a disabled textbox with the **ReadOnlyUI** property on the Customizer. **ReadOnlyUI** has two values: `Disabled` and `Label`.

```
<desDD:Customizer ID="Customizer1" runat="server"
   DenyEditRestrictionShowsReadOnly="true" ReadOnlyUI="Label" />
```

ReadOnly mode shows a User Control

A User Control can provide an alternative interface, such as showing text indicating the element is hidden and changing the style sheet of the containing column. DES defines the class `PeterBlum.DES.DynamicData.RestrictedFieldTemplate` to be used by any User Control that is intended to replace a Field Template in a restricted situation.

DES includes this User Control file: **~/DynamicData/Content/RestrictedField.ascx**. It is the default User Control, showing the text "n/a" and supplying style sheets to make the background grey. Feel free to edit it. The **RestrictedField.ascx** file includes documentation to assist you.

1. Create or modify an existing User Control that is based on `PeterBlum.DES.DynamicData.RestrictedFieldTemplate`. DES supplies **~/DynamicData/Content/RestrictedField.ascx**.

   Your page will use the same User Control for each restricted column. If you need your User Control to vary based on the Table, Column or Role, write code in the `Page_Load()` method to customize your User Control's content. The RestrictedFieldTemplate class supplies you with access to the MetaColumn object that was originally to appear in its **Column** property. **Column.Table** holds the MetaTable object.
2. Add the Customizer control to the page if its not already there. See "Using the Customizer Control".
3. Set the property **DenyViewRestrictionShowsAlternative** to `true` on the Customizer control.
4. If your UserControl does not use the path , assign the URL to your User Control in the **DenyViewRestrictionUserControlPath** property.

```
<desDD:Customizer ID="Customizer1" runat="server"
   DenyEditRestrictionShowsReadOnly="true"
   DenyViewRestrictionUserControlPath="~/DynamicData/Content/HideProducts.ascx" />
```

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

Customizing Automatic Scaffolding

By default, Automatic Scaffolding omits columns that are restricted. You may prefer to keep the column and let the DES.DynamicControl provide an alternative interface, as described in "Showing an alternative interface with a DES.DynamicControl".

Automatic Scaffolding uses the Customizer's **DenyEditRestrictionShowsReadOnly and DenyViewRestrictionShowsAlternative** to determine if it keeps the column or not. See "Showing an alternative interface with a DES.DynamicControl".

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

Using the RestrictionManager in your own code

The RestrictionManager class provides you with services to determine if a table or column needs security. You can generally access it with several static methods on the `PeterBlum.DES.DynamicData.DESMetaModel` class.

Click on any of these topics to jump to them:

- CanTakeActionOnTable method
- IdentifyTableRestrictions method
- CanTakeActionOnColumn method
- IdentifyColumnRestrictions method
- HasRole method

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

CanTakeActionOnTable method

Use the `DESMetaModel.CanTakeActionOnTable()` method to determine if specific restrictions apply to a table. It evaluates the TableRestrictionAttributes on the table's metadata. It is passed the MetaTable object for the table and a bitset of restrictions or the name of an action. It returns true if there are no matching restrictions and false if there are restrictions.

This is a static (or Shared in VB) method on `PeterBlum.DES.DynamicData.DESMetaModel`.

[C#]

```
static bool CanTakeActionOnTable(MetaTable pMetaTable,
  PeterBlum.DES.DynamicData.DenyAccess pRestrictions)
```

```
static bool CanTakeActionOnTable(MetaTable pMetaTable,
  string pAction)
```

[VB]

```
Shared Function CanTakeActionOnTable(ByVal pMetaTable As MetaTable, _
  ByVal pRestrictions As PeterBlum.DES.DynamicData.DenyAccess) As Boolean
```

```
Shared Function CanTakeActionOnTable(ByVal pMetaTable As MetaTable, _
  ByVal pAction As String) As Boolean
```

Parameters

pMetaTable

The table to evaluate.

pRestrictions

One or more restrictions. See "Using the DES.TableRestrictionAttribute" for the DenyAccess restriction values. If you supply multiple restrictions, at least one must be present on the TableRestrictionAttribute for this method to return `true`.

pAction

The Dynamic Data action to evaluate. They are "Details", "List", "Edit", "Insert", and "Delete".

Returns

When `true`, there were no restrictions found for the given table and restrictions. When `false`, at least one restriction matched to *pRestrictions*.

Example

[C#]

```
using DES=PeterBlum.DES.DynamicData;
...
if (DES.DESMetaModel.CanTakeActionOnTable(vMetaTable, DES.DenyAccess.Edit))
{
  // take action based on the restriction
}
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
...
If DES.DESMetaModel.CanTakeActionOnTable(vMetaTable, DES.DenyAccess.Edit) Then
  ' take action based on the restriction
End If
```

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

IdentifyTableRestrictions method

Use the `DESMetaModel.IdentifyTableRestrictions()` method to retrieve the user's restrictions on a table. It evaluates the TableRestrictionAttributes on the table's metadata. It is passed the MetaTable object for the table. It returns a bitset of restrictions or a value of `DenyAccess.None` if there are no restrictions.

This is a static (or Shared in VB) method on `PeterBlum.DES.DynamicData.DESMetaModel`.

[C#]

```
static bool IdentifyTableRestrictions(MetaTable pMetaTable)
```

[VB]

```
Shared Function IdentifyTableRestrictions(ByVal pMetaTable As MetaTable) _
  As Boolean
```

Parameters

pMetaTable

The table to evaluate.

Returns

A bitset of DenyAccess flags. See "Using the DES.TableRestrictionAttribute" for the DenyAccess restriction values.

First test if its value equals DenyAccess.None, which indicates no restrictions. To evaluate a specific restriction, use a bitset AND and see if the value matches the desired restriction.

Example

[C#]

```
using DES=PeterBlum.DES.DynamicData;
...
DES.DenyAccess vRestrictions = DES.DESMetaModel.IdentifyTableRestrictions(vMetaTable);
if ((vRestrictions & DES.DenyAccess.Edit) == DES.DenyAccess.Edit)
{
  // take action based on the restriction
}
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
...
Dim vRestrictions As DES.DenyAccess = _
  DES.DESMetaModel.IdentifyTableRestrictions(vMetaTable)
If (vRestrictions And DES.DenyAccess.Edit) = DES.DenyAccess.Edit Then
  ' take action based on the restriction
End If
```

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

CanTakeActionOnColumn method

Use the `DESMetaModel.CanTakeActionOnColumn()` method to determine if specific restrictions apply to a Column. It evaluates the ColumnRestrictionAttributes on the Column's metadata. It is passed the MetaColumn object for the Column and a bitset of restrictions or the name of an action. It returns `true` if there are no matching restrictions and `false` if there are restrictions.

This is a static (or Shared in VB) method on `PeterBlum.DES.DynamicData.DESMetaModel`.

[C#]

```
static bool CanTakeActionOnColumn(MetaColumn pMetaColumn,
  PeterBlum.DES.DynamicData.DenyAccess pRestrictions)
```

```
static bool CanTakeActionOnColumn(MetaColumn pMetaColumn,
  string pAction)
```

[VB]

```
Shared Function CanTakeActionOnColumn(ByVal pMetaColumn As MetaColumn, _
  ByVal pRestrictions As PeterBlum.DES.DynamicData.DenyAccess) As Boolean
```

```
Shared Function CanTakeActionOnColumn(ByVal pMetaColumn As MetaColumn, _
  ByVal pAction As String) As Boolean
```

Parameters

pMetaColumn

The Column to evaluate.

pRestrictions

One or more restrictions. See "Using the DES.ColumnRestrictionAttribute" for the DenyAccess restriction values. If you supply multiple restrictions, at least one must be present on the ColumnRestrictionAttribute for this method to return `true`.

pAction

The Dynamic Data action to evaluate. They are "Details", "List", "Edit", "Insert", and "Delete".

Returns

When `true`, there were no restrictions found for the given Column and restrictions. When `false`, at least one restriction matched to *pRestrictions*.

Example

[C#]

```
using DES=PeterBlum.DES.DynamicData;
...
if (DES.DESMetaModel.CanTakeActionOnColumn(vMetaColumn, DES.DenyAccess.Edit))
{
  // take action based on the restriction
}
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
...
If DES.DESMetaModel.CanTakeActionOnColumn(vMetaColumn, DES.DenyAccess.Edit) Then
  ' take action based on the restriction
End If
```

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

IdentifyColumnRestrictions method

Use the `DESMetaModel.IdentifyColumnRestrictions()` method to retrieve the user's restrictions on a Column. It evaluates the ColumnRestrictionAttributes on the Column's metadata. It is passed the MetaColumn object for the Column. It returns a bitset of restrictions or a value of `DenyAccess.None` if there are no restrictions.

This is a static (or Shared in VB) method on `PeterBlum.DES.DynamicData.DESMetaModel`.

[C#]

```
static bool IdentifyColumnRestrictions(MetaColumn pMetaColumn)
```

[VB]

```
Shared Function IdentifyColumnRestrictions(ByVal pMetaColumn As MetaColumn) _
  As Boolean
```

Parameters

pMetaColumn

The Column to evaluate.

Returns

A bitset of DenyAccess flags. See "Using the DES.ColumnRestrictionAttribute" for the DenyAccess restriction values.

First test if its value equals `DenyAccess.None`, which indicates no restrictions. To evaluate a specific restriction, use a bitset AND and see if the value matches the desired restriction.

Example

[C#]

```
using DES=PeterBlum.DES.DynamicData;
...
DES.DenyAccess vRestrictions =
  DES.DESMetaModel.IdentifyColumnRestrictions(vMetaColumn);
if ((vRestrictions & DES.DenyAccess.Edit) == DES.DenyAccess.Edit)
{
  // take action based on the restriction
}
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
...
Dim vRestrictions As DES.DenyAccess = _
  DES.DESMetaModel.IdentifyColumnRestrictions(vMetaColumn)
If (vRestrictions And DES.DenyAccess.Edit) = DES.DenyAccess.Edit Then
  ' take action based on the restriction
End If
```

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

HasRole method

Use the `DESMetaModel.HasRole()` method to determine if the current login is using one or more roles. It is passed a pipe delimited list of roles and either a MetaColumn or MetaTable object. It returns `true` if at least one role is associated with the current user and `false` if not.

This is a static (or Shared in VB) method on `PeterBlum.DES.DynamicData.DESMetaModel`.

[C#]

```
static bool HasRole(string pRoles, MetaColumn pMetaColumn)
static bool HasRole(string pRoles, MetaTable pMetaTable)
```

[VB]

```
Shared Function HasRole(ByVal pRoles As String, _
  ByVal pMetaColumn As MetaColumn) As Boolean
Shared Function HasRole(ByVal pRole As String, _
  ByVal pMetaTable As MetaTable) As Boolean
```

Parameters

pMetaColumn

A MetaColumn object, used to locate the DESMetaModel object.

pMetaTable

A MetaTable object, used to locate the DESMetaModel object.

pRoles

A pipe delimited list of one or more user roles. At least one must be found for this function to return true. It uses a case insensitive match.

Returns

When `true`, there was at least one role in *pRoles* found amongst the current user's roles. When `false`, there were no roles matching.

Example

[C#]

```
using DES=PeterBlum.DES.DynamicData;
...
if (DES.DESMetaModel.HasRoles("Admin|Supervisor", vMetaColumn))
{
  // take action based on the roles
}
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
...
If DES.DESMetaModel.HasRoles("Admin|Supervisor", vMetaColumn) Then
  ' take action based on the roles
End If
```

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

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

Web Form Development Overview

Building your web forms with ASP.NET Dynamic Data is different than when not using ASP.NET Dynamic Data. You no longer write business logic in the form, nor do you specify the controls that make up individual columns in your tables. That leaves you to do the following:

- Add a DataBound control, like GridView, ListView, DynamicFormView, DetailsView, FormView, or DynamicFormView. It will contain all controls and HTML related to your data-driven fields. *DES introduces DynamicListView and DynamicFormView along with their companion technology, Pattern Templates.*
    - Enable the control's "scaffolding" capability to let it generate the list of columns from your EntityTable Metadata classes or from code you add to your web form. See "Automatic Scaffolding Overview".
    - If you want to specify which columns are shown, you will not use scaffolding. Instead, you will use the DynamicControl, DynamicField (in the case of GridView and DetailsView), or DataFieldInPattern object (in the case of DynamicListView and DynamicFormView) where a column's data is displayed. Just by setting its **DataField** property to the column name, it will be ready with a default user interface.
    - Populate the DataBound control with any other controls, HTML, and style sheets to give it the desired appearance. When using Pattern Templates, the HTML and style sheets are modified in separate files.
- Add a DataSource control that is supported by ASP.NET Dynamic Data and works with the EntityType classes that are defined for you. Connect the DataSource to your DataBound control's **DataSourceID** property. See "Choosing a Technology for your Business Logic".
- Add DES's DynamicDataManager control and use its **Adapters** collection to let Dynamic Data know about the various controls it will be interacting with.
- Add DES's ValidationSummary control on forms that allow editing.

That's only 4 main controls (plus the contents within the DataBound control) and no coding!

Examples start on the next page.

Click on any of these topics to jump to them:

- Several ways to work: Choosing the right technique

Techniques:

- Building Applications with PageTemplates
- Building Applications without PageTemplates
- Pattern Templates: Reusable formatting for DynamicControls

Controls:

- DynamicDataManager Control and Adapters
- Customizer Control
- ColumnValidatorManager Control
- DynamicValidator Control
- The DynamicLabel Control

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

Several ways to work: Choosing the right technique

From the two examples above, you can see that there is more than one way to define the layout of your data-driven web forms. In fact, when you use Page Templates and scaffolding, you don't have to create any web forms! Here are some of the key features available to web form developers:

- **Scaffolding** – Let your business objects determine the exact list of columns to show. When a new column is added to your database, it automatically appears in your web form, unless the business logic marks that column (the property in the EntityTable Metadata class) with the ScaffoldColumnAttribute(false).

  Scaffolding is available on these DataBound controls: DynamicListView, DynamicFormView, GridView and DetailsView. If you use the FormView and ListView controls, use DynamicPattern control (described below) to support scaffolding.

  For details, see "Automatic Scaffolding Overview".

  > **Advantages**: Instantly creates the correct list of data driven controls. Reflects updates made to your database.
  >
  > **Disadvantages**: Limited control over the layout of the page. When you need to use a different list of columns than supplied by the business logic, you must either abandon it or implement the IPatternTemplateFieldGeneratorGuidance or IDataControlFieldGeneratorGuidance interface.

- **Page Templates** – Web forms that shipped with Dynamic Data and are located in the **/DynamicData/PageTemplates** folder. They contain all of the controls needed to deliver a Dynamic Data driven user interface. Since they utilize scaffolding, you will get an instant application built around your database. However, they have a predefined layout.

  To provide some additional flexibility, there are **Custom Page Templates**, where are basically the same web forms, except located in **/DynamicData/CustomPages**. You define them for specific tables. This allows you to explicitly define the column list, instead of using scaffolding. It also lets you switch from the inflexible layouts of GridView and DetailsView to the flexibility of ListView and FormView.

  For details, see "Building Applications with PageTemplates".

  > **Advantages**: Instantly creates an application.
  >
  > **Disadvantages**: The layout assumes that your table data is the primary feature of the page. Many times, DataBound controls share the page with other elements.
  >
  > To change the layout, you must use Custom Page Templates, which require additional setup. Once you have Custom Page Templates, any layout changes you make to the standard Page Templates may have to be also applied manually to the Custom Page Templates.

- **Web forms without Page Templates** – The introduction to this section shows the basics of creating web forms without Page Templates. You have to setup the 4 main controls: DataBound control, DataSource, DynamicDataManager, and ValidationSummary. You can use scaffolding or specify the data field names in DataFieldInPattern, DynamicControl, or DynamicField objects.

Control	Using Scaffolding	Layout
DetailsView	**AutoGenerateRows** propert	Fixed, table style
GridView	**AutoGenerateColumns** property	Fixed, table style
FormView	DynamicPattern control using **AutoScaffoldDataFields** property.	Flexible
ListView	DynamicPattern control using **AutoScaffoldDataFields** property.	Flexible
DynamicFormView	**AutoScaffoldDataFields** property	Flexible
DynamicListView	**AutoScaffoldDataFields** property	Flexible

  For details, see "Building Applications without PageTemplates".

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

Advantages: The most control you have of layout and available fields. If you use GridView and DetailsView, you can still use scaffolding.

Disadvantages: More work, as you have to setup all elements yourself. Take advantage of Pattern Templates (described next) to limit the impact and allow scaffolding even on FormView and ListView controls.

- **Pattern Templates** – Introduced in DES Dynamic Data, these are reusable UserControls that contain a block of HTML and DynamicControls. Their DynamicControls do not have their **DataField** properties assigned. Instead, you provide one or more column names in your DataBound control and they are applied at runtime.

  For example, here are two columns side-by-side, separated by "` `", in the user control named "TwoColumnsSideBySide.ascx".

```
<%@ Control Language="C#"
Inherits="PeterBlum.DES.DynamicData.DataFieldsPatternTemplate" %>

<desDD:DynamicLabel id="DL1" runat="server"
   AssociatedControlID="DynamicControl1" CssClass="LabelStyle" /> 
<desDD:DynamicControl id="DynamicControl1" runat="server"
   DataField="columnname1" /> 
<desDD:DynamicLabel id="DL2" runat="server"
   AssociatedControlID="DynamicControl2" CssClass="LabelStyle" /> 
<desDD:DynamicControl id="DynamicControl2" runat="server"
   DataField="columnname2" /><br />
```

  The **DynamicPattern** web control is used to specify a Pattern Template and its list of columns. Add it to the Template sections of FormView and ListView controls. Here is the DynamicPattern control specifying the TwoColumnsSideBySide.ascx Pattern Template:

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="TwoColumnsSideBySide" >
   <desDD:DataFieldInPattern DataField="columnname1" />
   <desDD:DataFieldInPattern DataField="columnname2" />
</desDD:DynamicPattern>
```

  For details, see "Pattern Templates: Reusable formatting for DynamicControls".

  Advantages: Helps you build free-form web forms more quickly and with a more consistent interface by reusing layouts. Supports scaffolding

  Disadvantages: Requires the extra work of creating free-form web forms.

EXAMPLES START ON THE NEXT PAGE

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

Example using DynamicFormView specifying individual field objects

```
<desDD:DynamicDataManager id="DDM1" other properties >
  <Adapters>
    <desDD:DynamicFormViewAdapter DataBoundControlID="DynamicFormView1"
      ValidationSummaryControlID="ValidationSummary1"
      SupportsEditMode="true" SupportsInsertMode="true" />
  </Adapters>
</desDD:DynamicDataManager>

<des:ValidationSummary id="ValidationSummary1" other properties />
<desDD:DynamicFormView id="DynamicFormView1" DataSourceID="LinqDataSource1"
  other properties >
<ItemsInPattern>
  <desDD:DataFieldInPattern DataField="ProductName" />
  <desDD:DataFieldInPattern DataField="Price" />
  <desDD:DataFieldListInPattern NamedPart="LinkButtons" />
</ItemsInPattern>
</desDD:DynamicFormView>

<asp:LinqDataSource id="LinqDataSource1" EnableUpdate="true" EnableInsert="true"
  other properties />
```

Example using DynamicListView with automatic scaffolding

```
<desDD:DynamicDataManager id="DDM1" other properties >
  <Adapters>
    <desDD:DynamicListViewAdapters DataBoundControlID="DynamicListView1"
      ValidationSummaryControlID="ValidationSummary1"
      SupportsEditMode="true" SupportsInsertMode="true" />
  </Adapters>
</desDD:DynamicDataManager>

<des:ValidationSummary id="ValidationSummary1" other properties />
<asp:DynamicListView id="DynamicListView1" DataSourceID="LinqDataSource1"
  other properties >
</asp:DynamicListView>

<asp:LinqDataSource id="LinqDataSource1" EnableUpdate="true" EnableInsert="true"
  other properties />
```

EXAMPLES CONTINUE ON THE NEXT PAGE

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

Example using DetailsView specifying individual field objects

```
<desDD:DynamicDataManager id="DDM1" other properties >
  <Adapters>
    <desDD:DetailsViewAdapter DataBoundControlID="DetailsView1"
      ValidationSummaryControlID="ValidationSummary1"
      SupportsEditMode="true" SupportsInsertMode="true" />
  </Adapters>
</desDD:DynamicDataManager>

<des:ValidationSummary id="ValidationSummary1" other properties />
<asp:DetailsView id="DetailsView1" DataSourceID="LinqDataSource1" other properties >
<Columns>
  <desDD:DynamicField DataField="columnname1" />
  <desDD:DynamicField DataField="columnname2" />
  <asp:CommandField properties />
</Columns>
</asp:DetailsView>

<asp:LinqDataSource id="LinqDataSource1" EnableUpdate="true" EnableInsert="true"
  other properties />
```

Example using FormView with DynamicControl objects and HTML formatting

```
<desDD:DynamicDataManager id="DDM1" other properties >
  <Adapters>
    <desDD:FormViewAdapters DataBoundControlID="FormView1"
      ValidationSummaryControlID="ValidationSummary1"
      SupportsEditMode="true" SupportsInsertMode="true" />
  </Adapters>
</desDD:DynamicDataManager>

<des:ValidationSummary id="ValidationSummary1" other properties />
<asp:FormView id="FormView1" DataSourceID="LinqDataSource1" other properties >
<ItemTemplate>
  <desDD:DynamicLabel id="DL1" runat="server"
    AssociatedControlID="DynamicControl1" CssClass="LabelStyle" /> 
  <desDD:DynamicControl id="DynamicControl1" runat="server"
    DataField="columnname1" /><br />
  <desDD:DynamicLabel id="DL2" runat="server"
    AssociatedControlID="DynamicControl2" CssClass="LabelStyle" /> 
  <desDD:DynamicControl id="DynamicControl2" runat="server"
    DataField="columnname2" /><br />
  <des:Button id="EditButton" runat="server" CommandName="Edit"
    Text="Edit" CausesValidation="false" /> 
  <des:Button id="NewButton" runat="server" CommandName="New"
    Text="New Record" CausesValidation="false" /><br />
</ItemTemplate>
</asp:FormView>

<asp:LinqDataSource id="LinqDataSource1" EnableUpdate="true" EnableInsert="true"
  other properties />
```

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

Building Applications with PageTemplates

Page Templates are web forms that provide a very quick way to deliver an application that explores and edits a database. These web forms that shipped with Dynamic Data and are located in the **/DynamicData/PageTemplates** folder. They contain all of the controls needed to deliver a Dynamic Data driven user interface. Since they utilize scaffolding, you will get an instant application built around your database. However, they have a predefined layout.

While its setup is quick, it has a specific implementation that may not suit your needs. If it doesn't, see "Building Applications without PageTemplates".

Click on any of these topics to jump to them:

- Automatic Scaffolding Overview
- Files in the PageTemplates System
- Steps for using PageTemplates
- Customizing the standard PageTemplates
- Creating Table-Specific PageTemplates
- Customizing Automatic Scaffolding
- Customizing the Master Page
- Customizing the Entry Point Page
- Changing the Default paths for PageTemplates

- DES.DynamicControl: The heart of Dynamic Data
- DES.DynamicField: Using Dynamic Data in GridView and DetailsView
- DynamicDataManager Control and Adapters
- Customizer Control
- ColumnValidatorManager Control
- DynamicValidator Control
- The DynamicLabel Control

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

Automatic Scaffolding Overview

Automatic Scaffolding defines the data fields shown tables in DataBound controls so you don't have define them manually. For example, the GridView control will populate its **Columns** property with DynamicFields when you set **AutoGenerateColumns** to `true`. Similarly, the DynamicFormView and DynamicListView controls will populate their **ItemsInPattern** collection with DataFieldInPattern objects when its **ItemsInPattern** collection is empty or you set the **AutoScaffoldDataFields** property to `True`.

Automatic Scaffolding determines the list of fields and its order from the properties on the EntityTable classes. It creates them in the order found, omitting any that has a ScaffoldColumnAttribute(false). When no ScaffoldColumnAttribute is specified, the following conditions determine the column is being scaffolded by evaluating the MetaColumn's properties in this order.

- Kept: A Field Template name is specified in the UIHintAttribute.
- Omitted: **MetaColumn.IsForeignKeyComponent** = `true` (Part of a ForeignKeyColumn)
- Omitted: **MetaColumn.IsGenerated** = `true` (The column's data is generated by the database)
- Kept: **MetaColumn.IsPrimaryKey** = `true` (The column is a primary key)
- Omitted: **MetaColumn.IsCustomProperty** = `true` (The column was created in the EntityTable class, but does not exist in the database)
- Kept: MetaColumnType is one of these: string, char, integer, floating point, bool, DateTime, TimeSpan, or DateTimeOffset.
- Omitted: Anything not listed above.

Click on any of these topics to jump to them:

- PageTemplates and Automatic Scaffolding
- Overcoming the limitations of Automatic Scaffolding
  - Tools to customize Automatic Scaffolding

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

PageTemplates and Automatic Scaffolding

The Page Templates system can create an application to explore and edit your database very quickly through the Automatic Scaffolding feature. The Page Templates supply the overall look of the web form. They use a DynamicListView or GridView control for list oriented browsing and a DynamicFormView or DetailsView control for single record viewing and editing. The DataBound controls do not have a specified table or list of columns to show. Instead, ASP.NET URL Routing delivers the table while Automatic Scaffolding provides the columns.

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

Overcoming the limitations of Automatic Scaffolding

Automatic Scaffolding has some limitations that do not make it useful in all cases:

- You can omit columns but you must use the ScaffoldColumnAttribute in the EntityTable Metadata classes. As a result, each column has a single state: visible or not. If you want a list to show a different set of columns from the detail, you must declare the list of fields within the DataBound control instead of using scaffolding. (See "Creating Table-Specific PageTemplates")

  In addition, the ScaffoldColumnAttribute breaks the goal of separation of concerns because it controls the user interface.

  Solution: DES Dynamic Data resolves this by introducing the IPatternTemplateFieldGeneratorGuidance and IDataControlFieldGeneratorGuidance interfaces where you programmatically determine if a column is included or not, with code in your Master Page.

- The column ordering is determined by the order of properties defined in the EntityTable classes. Often that ordering does not reflect what you want in the user interface.

  Solution: DES Dynamic Data resolves this in two ways: by expanding the ScaffoldColumnAttribute to support positioning rules and through the IPatternTemplateFieldGeneratorGuidance and IDataControlFieldGeneratorGuidance interfaces, where you can call methods that change the order.

- It can only define DynamicField objects. Perhaps you want several columns combined into a single cell with a TemplateField object or have subclassed DynamicField.

  Solution: DES Dynamic Data resolves this through the IPatternTemplateFieldGeneratorGuidance and IDataControlFieldGeneratorGuidance interfaces, allowing you to create any type of DataControlField object (for GridView and DetailsView) or ItemInPattern object (for DynamicPattern, DynamicListView, and DynamicFormView).

Tools to customize Automatic Scaffolding

The IPatternTemplateFieldGeneratorGuidance and IDataControlFieldGeneratorGuidance interfaces are central to customizing scaffolding. By implementing one on your Master Page, you can:

- Add and remove field objects that are generated automatically from the metadata.
- Change the ordering of the field objects.
- Completely ignore the built in generation of field objects and create the exact list you need, based on the DataBound Control's usage (List, Details, Insert, Edit), mode (ReadOnly, Edit, Insert), or style (List oriented vs single record oriented).
- With IPatternTemplateFieldGeneratorGuidance, you can define alternative ItemInPattern objects (like TemplateNotInPattern and DataFieldListInPattern). If you subclass from DataFieldInPattern, you can define your subclass.
- With IDataControlFieldGeneratorGuidance, you can define alternative DataControlFields (like TemplateField and ButtonField). If you subclass DynamicField, you can define your subclass.
- As you create the field object, set Field Template specific properties on it to customize that Field Template.
- Override several of the hardcoded rules found in the native FieldGenerator class.

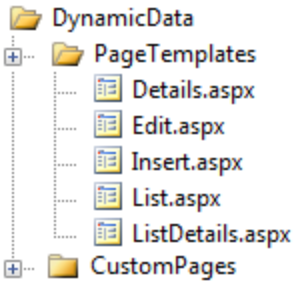
See "Customizing Automatic Scaffolding".

The DES.ScaffoldColumnAttribute provides another way to control some of the Automatic Scaffolding. Since this attribute is part of the EntityTable Metadata class, it should not be used if you want to separate business logic from user interface. If you use it, it can determine if the column appears or not. It can also determine the position of the column. See "ScaffoldColumnAttribute: Determining if Scaffolding adds a Column".

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

Files in the PageTemplates System

Here are the key elements of the Page Templates system:



- Web forms in the **[web application]/DynamicData/PageTemplates** folder – The forms handle each of the 5 database oriented cases:
  - List.aspx – Uses a DynamicListView to list rows in a single table and hyperlink to other tables. Provides commands for Edit, Delete, Insert, Select (highlight a row), and Details (show the record in read-only mode).
  - Details.aspx – Uses a DynamicFormView to list the columns of a single record in a read only mode. Provides hyperlinks to other tables where the column uses a ForeignKey. Provides commands for Edit, Delete, and return to the List.
  - Edit.aspx – Uses a DynamicFormView to edit the columns of a single record. It is used in Edit mode, offering buttons to Update (save) and Cancel the edits.
  - Insert.aspx – Uses a DynamicFormView to insert a record. It is used in Insert mode, offering buttons to Insert (save) and Cancel the entry.
  - ListDetails.aspx – An alternative user interface to the separate forms for list, insert, edit, above. It includes both the DynamicListView and DynamicFormView on one form. The user will always have access to the list even when working in the DetailsView to insert or edit.

  These web forms support every table, as they let Automatic Scaffolding create their field objects from the EntityTable Classes.

  See "Customizing the standard PageTemplates".

- Web forms in the **[web application]/DynamicData/CustomPages** folder – Forms that override the default forms listed above on a table-by-table basis. You simply create a subfolder with the name of the table, such as "Products", and a copy of the web forms taken from the **[web application]/DynamicData /PageTemplates** folder. You usually turn off Automatic Scaffolding and define a specific list of fields.

  See "Creating Table-Specific PageTemplates".

  You can avoid using the CustomPages folder by Customizing Automatic Scaffolding.

- ASP.NET URL Routing – Your `Application_Start()` method (in **global.asax**) defines `System.Web.DynamicData.DynamicDataRoute` objects that determine which of these forms to use given the action. Actions are List, Details, Edit, and Insert. See "Mapping PageTemplates to Actions" and "Set up ASP.NET URL Routing".

- Master Page – Each Page Template refers to a Master Page to provide a uniform user interface around the DataBound controls. It supplies style sheets to the content pages too. The default Page Templates are connected to Master Page file shipped with ASP.NET Dynamic Data, called **Site.master**, located in the **[web application]** folder.

  A typical Master Page for Page Templates has these features:

  - Loads the style sheet file **Site.css**, which is included with ASP.NET Dynamic Data. DES Dynamic Data has additional style sheet files where are automatically added as needed.
  - Supports ASP.NET AJAX with the ScriptManager control. The Page Templates which define the content pages usually has the UpdatePanel surrounding the DataBound control and ValidationSummary.
  - After adding DES to your web application, it will contain the DES PageManager control.
  - Optionally, you can add the Customizer control here.

  See "Customizing the Master Page".

- Entry page – A web form that greets the user with a list of available tables. ASP.NET Dynamic Data ships with one called **Default.aspx** in your **[web application]** folder.

  See "Customizing the Entry Point Page".

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

Steps for using PageTemplates

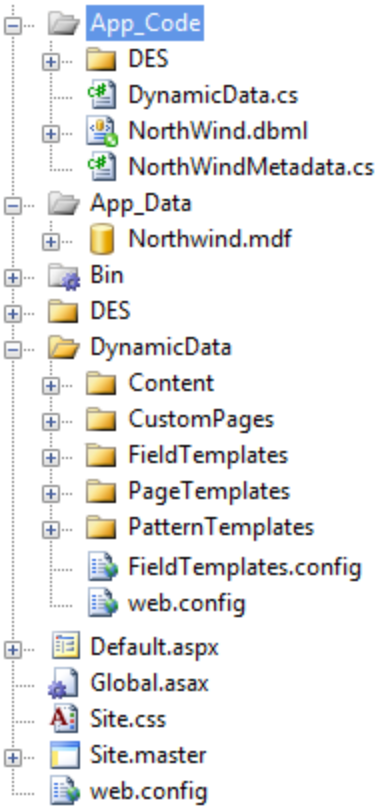
These steps provide a big picture of how to use the Page Templates system. Remember that Page Templates are selected by using the ASP.NET Url Routing features and are complete web forms with a Master Page. If you do not want some of these features, see "Building Applications without PageTemplates".

At this point, you should have already taken these actions:

- Added ASP.NET Dynamic Data to your web application. See "Adding ASP.NET Dynamic Data".
- Adding DES Dynamic Data to your web application. See "Adding DES Dynamic Data".
- Setup Dynamic Data globals in `Application_Start()`. See "Initializing Dynamic Data at application startup".
- Created the Data Context and EntityTable classes. See "Creating EntityTable Classes".

Quite often, the Page Templates folder has the right setup but the Master Page and its style sheet are not right for your application. So we'll start with the Master Page.

These steps ask you 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. Each Page Template points to a Master Page, which defaults to **Site.Master** in the web application root folder. Here are some common tasks.
   - Renaming the Master Page
   - Change the appearance of the Master Page
   - Change the style sheet file
2. Determine whether you want to use individual pages for the basic actions (List, Details, Edit and Insert) or the **ListDetails.aspx** page which handles all actions. You edit the DynamicDataRoute objects established in the `Application_Start()` method of **global.asax**.

   See "Mapping PageTemplates to Actions".
3. By default, Automatic Scaffolding will deliver a list of fields. When you need to specify or order the list of fields for a table, there are several approaches available:
   a. Use the DES.ScaffoldColumnAttribute on a column's property in the EntityTable Metadata Class to determine its inclusion and ordering.

      See "ScaffoldColumnAttribute: Determining if Scaffolding adds a Column"

   b. Implement the IPatternTemplateFieldGeneratorGuidance interface to programmatically define the columns shown in the DataBound controls.

      See "Customizing Automatic Scaffolding".

   c. Create a Page Template web form and add it to the **DynamicData/CustomPages** folder under a folder whose name matches the table's name.

      That Page Template may mimic all aspects of the same named file in the **PageTemplates** folder, but the DataBound controls will have **AutoGenerateFields**=`false` and their **Fields** collection populated with field objects for the desired columns.

      See "Creating Table-Specific PageTemplates".

CONTINUED ON THE NEXT PAGE

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

4. If you want to omit any table found in the Data Context when using the Scaffolding feature, add the ScaffoldTableAttribute to the EntityTable Class.

   See “SortableColumnAttribute: Determining ”.

5. The entry web form, **Default.aspx**, may not have the right user interface or even be desired. Here are some common tasks.

   - Renaming the Entry Point Page
   - Going to a specific table (alternative to an Entry Point Page)

6. Customize the web forms in the **PageTemplates** folder. See “Customizing the standard PageTemplates”.

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

Customizing the standard PageTemplates

This section discusses the elements of Page Templates. It does not provide directions for defining the list of fields shown in the DataBound controls. For that, please see step 3 in the previous section.

Click on any of these topics to jump to them:

- Before you start
- Elements of a PageTemplate
- Setting Properties on your Data Source
- Setting Properties on your DataBound control
- Adding Buttons to your DataBound control
- Redirecting to another PageTemplate on a button click

Related Topics:

- Creating Table-Specific PageTemplates
- Customizing Automatic Scaffolding
- Customizing the Master Page
- Customizing the Entry Point Page
- Setting up the DataBound Control at Runtime
- DynamicDataManager Control and Adapters

Before you start

The standard Page Templates, located in the **[web application]/DynamicData/PageTemplates** folder are used when there is no Page Template corresponding to the requested table in **[web application]/DynamicData/ CustomPages/[tablename]** folder. As a result, any edits made to these files will be widely seen. In addition, since edits made here don't affect the Page Templates in the **CustomPages** folder, consider applying your changes in both places.

See also "Creating Table-Specific PageTemplates".

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

Elements of a PageTemplate

Each Page Template has a specific job, based on the action requesting the page. (See "Mapping PageTemplates to Actions".) **List.aspx** always shows a list of records. **Details.aspx**, **Insert.aspx**, and **Edit.aspx** show a single record, using readonly, insert, and edit modes respectively. **ListDetails.aspx** has both list and single record elements with all editing modes.

Each Page Template should have the following controls. Others are allowed as well.

Control	Usage
DynamicDataManager	Use the one supplied by DES. It connects the DataBound control, ValidationSummary, and other controls to Dynamic Data. See "DynamicDataManager Control and Adapters".
DataSource	Use a specialized DataSource for your situation: LinqDataSource, EntityDataSource, or another created for use with Dynamic Data. See "Setting Properties on your Data Source" for datasource properties.
DataBound Control	For lists, the DynamicListView control is typically used. GridView is an alternative. When creating a Page Template within the CustomPages folder, you may consider the ListView although the DynamicListView often is a better choice. For single record, the DynamicFormView control is typically used. DetailsView is an alternative. When creating a Page Template within the CustomPages folder, you may consider the FormView although the DynamicFormView often is a better choice. See "Setting Properties on your DataBound control" for key properties on DataBound controls.
ValidationSummary	Use DES's ValidationSummary. It is not needed on Details.aspx because no editing happens there. You can have one or two on ListDetails.aspx. The DES DynamicDataManager control sets its validation group.
Filtering	Optional. Used with lists to change the criteria of data. It guides the queries performed by the DataSource. ASP.NET Dynamic Data provides the FilterRepeater. Additional resources: ➢ "Extending the FilterRepeater" – by Marcin Dobosz. ➢ AdvancedFilterRepeater in Dynamic Data Futures.
UpdatePanel and ScriptManagerProxy	Optional. Provides AJAX updates around the DataBound control, ValidationSummary, and Filtering controls.

SAMPLES ARE ON THE NEXT TWO PAGES

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

Example: Edit.aspx

```
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
    <h2>Edit entry from table <%= table.DisplayName %></h2>
    <asp:ScriptManagerProxy runat="server" ID="ScriptManagerProxy1" />

    <desDD:DynamicDataManager ID="DynamicDataManager1" runat="server"
       AutoLoadForeignKeys="true" >
       <Adapters>
          <desDD:DynamicFormViewAdapter DataBoundControlID="RecordDetails"
             ValidationSummaryControlID="ValidationSummary1"
             SupportsEditMode="true" SupportsInsertMode="false"
             CommandsUseRouting="true"
             PostUpdateAction="List" PostCancelAction="List" />
       </Adapters>
    </desDD:DynamicDataManager>

    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
      <ContentTemplate>

          <des:ValidationSummary ID="ValidationSummary1" runat="server"
              InAJAXUpdate="True"
              HeaderText="List of validation errors" RelatedControlID="AfterValSum"
              RelatedControl_InvisiblePreservesSpace="false"  />
          <des:LocalizableLabel ID="AfterValSum" runat="server" >
          <br /><br /></des:LocalizableLabel>

          <desDD:DynamicFormView ID="RecordDetails" runat="server"
            DataSourceID="RecordDetailsDataSource" DefaultMode="Edit"
            AutoGenerateFields="true" AutoGenerateButtons="Last" >
          </desDD:DynamicFormView>

          <asp:LinqDataSource ID="RecordDetailsDataSource" runat="server"
            EnableUpdate="true">
              <WhereParameters>
                  <asp:DynamicQueryStringParameter />
              </WhereParameters>
          </asp:LinqDataSource>

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

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

Example: List.aspx

```
<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1"
Runat="Server">
    <h2><%= table.DisplayName%></h2>
    <asp:ScriptManagerProxy runat="server" ID="ScriptManagerProxy1" />

    <desDD:DynamicDataManager ID="DynamicDataManager1" runat="server"
         AutoLoadForeignKeys="true" >
       <Adapters>
          <desDD:DynamicListViewAdapter DataBoundControlID="RecordsList"
             ValidationSummaryControlID="ValidationSummary1"
             SupportsEditMode="True" SupportsInsertMode="False"
             CommandsUseRouting="true" />
       </Adapters>
    </desDD:DynamicDataManager>

    <asp:UpdatePanel ID="UpdatePanel1" runat="server">
      <ContentTemplate>

          <des:ValidationSummary ID="ValidationSummary1" runat="server"
              InAJAXUpdate="True"
              HeaderText="List of validation errors" RelatedControlID="AfterValSum"
              RelatedControl_InvisiblePreservesSpace="false"  />
          <des:LocalizableLabel ID="AfterValSum" runat="server" >
          <br /><br /></des:LocalizableLabel>

          <asp:FilterRepeater ID="FilterRepeater" runat="server">
              <ItemTemplate>
                  <asp:Label ID="Label1" runat="server"
                    Text='<%# Eval("DisplayName") %>'
                    AssociatedControlID="DynamicFilter$DropDownList1" />
                  <asp:DynamicFilter runat="server" ID="DynamicFilter"
                    OnSelectedIndexChanged="OnFilterSelectedIndexChanged" />
              </ItemTemplate>
              <FooterTemplate><br /><br /></FooterTemplate>
          </asp:FilterRepeater>

          <desDD:DynamicListView ID="RecordsList" runat="server"
             DataSourceID="RecordsListDataSource"
             AutoGenerateButtons="First"
             OpeningTagStyle-CssClass="GridViewTable" >
              <EmptyDataTemplate>
                  There are currently no items in this table.
              </EmptyDataTemplate>
          </desDD:DynamicListView>
          <desDD:DynamicControlPanel ID="DynamicControlPanel1" runat="server"
             PageSize="10" PagedControlID="RecordsList">
             <Fields>
                <desDD:PreviousPagerField />
                <desDD:InfoPagerField />
                <desDD:NextPagerField />
             </Fields>
          </desDD:DynamicControlPanel>
```

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

```
          <asp:LinqDataSource ID="RecordsListDataSource" runat="server"
             EnableDelete="true">
              <WhereParameters>
                  <asp:DynamicControlParameter ControlID="FilterRepeater" />
              </WhereParameters>
          </asp:LinqDataSource>
      </ContentTemplate>
    </asp:UpdatePanel>
    <br />

   <div class="bottomhyperlink">
      <asp:HyperLink ID="InsertHyperLink" runat="server">
       <img id="Img1" runat="server" src="~/DynamicData/Content/Images/plus.gif"
          alt="Insert new item" />
      Insert new item</asp:HyperLink>
   </div>

</asp:Content>
```

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

Setting Properties on your Data Source

All DataSources that are supported by ASP.NET Dynamic Data must implement `System.Web.DynamicData.IDynamicDataSource`. LinqDataSource, EntityDataSource, and IDynamicDataSource have most of following properties for your consideration:

- **ContextTypeName** (string) – It will automatically be set to the the Data Context Type that you assigned to the MetaModel. You should leave it alone in standard Page Templates. It must be set in pages that aren't loaded by the ASP.NET URL Routing feature using a DynamicDataRoute because when the DynamicDataRoute is involved, it contains the MetaModel.
- **TableName** or **EntitySetName** (string) – The name of the table. Only assign it when using **ContextTypeName**. Leave it alone in standard Page Templates as the Routing feature determines which table should be assigned.
- **EnableInsert** (Boolean) – When `true`, support Insert mode. It should be `true` on **Insert.aspx** and for the DataSource supporting the DetailsView on **ListDetails.aspx**. If you edit List.aspx to support Insert mode in the DynamicListView, it should be `true`.

  When `false` and you attempt to insert, an exception will be thrown.
- **EnableUpdate** (Boolean) – When `true`, support Edit mode. It should be `true` on **Edit.aspx** and for the DataSource supporting the DetailsView on **ListDetails.aspx**. If you edit List.aspx to support Edit mode in the DynamicListView, it should be `true`.

  When `false` and you attempt to save an existing record, an exception will be thrown.
- **EnableDelete** (Boolean) – When `true`, support the Delete command. It should be `true` on any form with a Delete command.

  When `false` and you attempt to delete, an exception will be thrown.
- **AutoGenerateWhereClause** (Boolean) – Gets or sets a value that indicates whether the DataSource control dynamically creates a **Where** clause based on values defined in the **WhereParameters** collection.

  See http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.linqdatasource.autogeneratewhereclause.aspx.
- **WhereParameters** (System.Web.UI.WebControls.ParameterCollection) – Gets the collection of parameters that is used to create the Where clause. The Where clause is the selection criteria against the table. For details, see http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.linqdatasource.whereparameters.aspx.

  ASP.NET Dynamic Data provides two classes to make it easier to setup the WhereParameters.

  - System.Web.DynamicData.DynamicQueryStringParameter – Use it when using the URL to specify the action, table, and optionally specific record. It converts the URL patterns assigned to the DynamicDataRoute class into the Where clause. It takes no parameters.

    ```
    <asp:DynamicQueryStringParameter />
    ```

    Use this on the DataSources of Page Templates associated with DetailsViews and FormsViews.
  - System.Web.DynamicData.DynamicControlParameter – Use it when using the FilterRepeater or another control that delivers the Where clause by implementing `System.Web.DynamicData.IWhereParametersProvider`. Specify the control's ID in the ControlID property.

    ```
    <asp:DynamicControlParameter ControlID="FilterRepeater1"/>
    ```

    Use this on the DataSources of Page Templates that have a FilteredRepeater.

For other properties, see LinqDataSource and EntityDataSource.

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

Setting Properties on your DataBound control

There are a variety of DataBound controls that support list and single record styles. ASP.NET includes GridView and ListView for lists; DetailsView and FormView for single records. DES adds (and recommends) DynamicListView and DynamicFormView. There are also third party grids which must subclass from `System.Web.UI.WebControls.DataBoundControl` to be supported by ASP.NET Dynamic Data. (Telerik RadGrid for ASP.NET AJAX is an example.)

Properties

There are many properties on each of these controls. Only a few are associated with ASP.NET Dynamic Data. They are covered here.

- **DataSourceID** (String) – The controlID of the DataSource control. Either this or the **DataSource** property must be assigned. The DataSource class must implement `System.Web.DynamicData.IDynamicDataSource`.
- **DataSource** (object) – An alternative to **DataSourceID**. Assign a reference to the DataSource control's object programmatically.
- **Mode** or **DefaultMode** (enum System.Web.UI.WebControls.DataBoundControlMode) – Identifies the control's mode: read only, edit or insert. The initial mode setting depends on the Page Template:

PageTemplate	Mode
List.aspx	Initially not set. Actions that set the EditIndex property automatically adjust the mode.
Details.aspx	DefaultMode = ReadOnly
Edit.aspx	DefaultMode = Edit
Insert.aspx	DefaultMode = Insert
ListDetails.aspx	For the list, initially not set. For the DetailsView or FormView, DefaultMode = ReadOnly. Use built in commands to change it automatically. Various actions on the list control should call DetailsView.ChangeMode(DataBoundControlMode.ReadOnly)

- **AutoScaffoldDataFields** (Boolean) – *DynamicListView and DynamicFormView controls only*. Gets or sets a value indicating whether each field in the data source is generated using Automatic Scaffolding. It defaults to `WhenEmpty` which populates the **ItemsInPattern** collection when its empty. If you add DataFieldInPattern objects to the **ItemsInPattern** collection, set it to `True` only if you also want scaffolded data fields.
- **AutoGenerateRows** (Boolean) – *DetailsView control only*. Gets or sets a value indicating whether each field in the data source is generated using Automatic Scaffolding. It defaults to `true`. If you add DynamicField controls to the **Rows** collection, set it to `false`.
- **AutoGenerateColumns** (Boolean) – *GridView control only*. Gets or sets a value indicating whether each field in the data source is generated using Automatic Scaffolding. It defaults to `true`. If you add DynamicField controls to the **Columns** collection, set it to `false`.

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

Adding Buttons to your DataBound control

The Automatic Scaffolding feature does not know how to add buttons that are essential in an editing interface, such as Edit, Save, and Cancel. You need to take advantage of features on the DataBound controls to add buttons.

Click on any of these topics to jump to them:

- Automatic generation: Let the DataBound control add them for you
- Declaring the buttons within the list of fields
- Using actual Button controls

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

Automatic generation: Let the DataBound control add them for you

Most DataBound controls have properties that create buttons for you.

DataBound Control	How to show the buttons
DynamicListView	**AutoGenerateButtons** = `true` The appearance is determined by Pattern Templates. There are two predefined, "Buttons" and "LinkButtons". "LinkButtons" is the default. Change the type in the **ButtonsNamedPart** property. Further customization can be done within the Pattern Template files ("ButtonsForList", "LinkButtonsForList").
GridView	See "Using Properties on the GridView and DetailsView".
ListView	The ListView does not support automatic generation of buttons. You must explicitly declare the buttons within its ItemTemplates. Use the DynamicButtons control to simplified this. See "Declaring the buttons within the list of fields".
DynamicFormView	**AutoGenerateButtons** = `true` The appearance is determined by Pattern Templates. There are two predefined, "Buttons" and "LinkButtons". "LinkButtons" is the default. Change the type in the **ButtonsNamedPart** property. Further customization can be done within the Pattern Template files ("DetailsViewButtons", "DetailsViewLinkButtons", "ButtonsForSingle", or "LinkButtonsForSingle").
DetailsView	See "Using Properties on the GridView and DetailsView".
FormView	The FormView does not support automatic generation of buttons. You must explicitly declare the buttons within its ItemTemplates. Use the DynamicButtons control to simplified this. See "Declaring the buttons within the list of fields".

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

Using Properties on the GridView and DetailsView

On the DetailsView and ListView, you have these properties that when `true`, add one or more buttons. However, they should only be used when the page will postback to take the action. If the button requires redirecting to another Page Template, see "Redirecting to another PageTemplate ".

Property	Buttons added
AutoGenerateEditButton	Edit button while in ReadOnly mode. When clicked, it switches to Edit mode. Update and Cancel buttons while in Edit mode. Set this to `true` in Edit.aspx and DetailsView in ListDetails.aspx.
AutoGenerateInsertButton	New button while in ReadOnly mode. When clicked, it switches to Insert mode. Insert and Cancel buttons while in Insert mode. Set this to `true` in Insert.aspx and DetailsView in ListDetails.aspx.
AutoGenerateDeleteButton	Delete button while in ReadOnly mode. When clicked, it deletes the record. Set this to `true` in Details.aspx and DetailsView in ListDetails.aspx.
AutoGenerateSelectButton	*GridView Only.* Select button for rows in ReadOnly mode. When clicked, it shows a DetailView either using Details.aspx or the DetailsView in ListDetails.asx. Set this to `true` in GridView in ListDetails.aspx.

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

Declaring the buttons within the list of fields

Sometimes the positioning or features of the automatic button generation don't fit your needs. You can declare the buttons more explicitly within the list of fields declared for the DataBound control. For instance, if you want buttons to appear before your data fields, do the following prior to the list of data fields.

With this approach, you can set properties, such as the text of the buttons, or their visibility.

DataBound Control	How to show the buttons

DynamicListView	**AutoGenerateButtons** = `None`
	Within the **ItemsInPattern** property, add a ButtonsInPattern object where you want the buttons.
	Assign the **NamedPart** property to "Buttons" or "LinkButtons".
	The Pattern Templates have numerous properties. See "Passing properties to be used by the PatternTemplate's own code". They can be declared as properties on the ButtonsInPattern object, such as the **ShowSelectButton** property shown here.
	`<ItemsInPattern>`
	`  <desDD:DataFieldListInPattern NamedPart="Buttons"`
	`  ShowSelectButton="True"/>`
	`</ItemsInPattern>`
	You can also use a TemplateNotInPattern object to create your buttons interface. It can host actual buttons or a DynamicButtons control.

GridView	Add the CommandField to the **Columns** property. In that case, set its **ShowEditButton**, **ShowInsertButton**, **ShowDeleteButton**, **ShowSelectButton**, and **ShowCancelButton** properties to `false`. Alternatively, use the ButtonField class, defining the **CommandName** property.

ListView	Use the DynamicButtons control to the various ItemTemplates where you want the buttons.
	`<ItemTemplate>`
	`  <desDD:DynamicButtons id="ReadOnlyButtons" runat="server" />`
	`</ItemTemplate>`
	DynamicButtons inside a PatternTemplate
	If you have a Pattern Template that already uses this control to establish the formatting, add a DynamicPattern control.
	Assign the **PatternTemplateName** property on DynamicPattern to "ButtonsForList", "LinkButtonsForLink" , or a customized version of those Pattern Templates.
	Set the **Mode** property to "ReadOnly", "Edit" or "Insert" depending on the type of ItemTemplate.
	`<ItemTemplate>`
	`  <desDD:DynamicPattern id="ReadOnlyFields" runat="server"`
	`  PatternTemplateName="ButtonsForList" Mode="ReadOnly" >`
	`  </desDD:DynamicPattern>`
	`</ItemTemplate>`

DynamicFormView	Same as DynamicListView, above.

CONTINUED ON THE NEXT PAGE

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

DataBound Control	How to show the buttons
DetailsView	Add the CommandField to the **Rows** property. In that case, set its **ShowEditButton**, **ShowInsertButton**, **ShowDeleteButton**, **ShowSelectButton**, and **ShowCancelButton** properties to `false`. Alternatively, use the ButtonField class, defining the **CommandName** property.

FormView	Use the DynamicButtons control to the various ItemTemplates where you want the buttons.
	`<ItemTemplate>`
	`  <desDD:DynamicButtons id="ReadOnlyButtons" runat="server" />`
	`</ItemTemplate>`
	
	DynamicButtons inside a PatternTemplate
	
	If you have a Pattern Template that already uses this control to establish the formatting, add a DynamicPattern control.
	
	Assign the **PatternTemplateName** property on DynamicPattern to "DetailsViewButtons" or "DetailsLinkButtons" or a customized version of those Pattern Templates.
	
	Set the **Mode** property to "ReadOnly", "Edit" or "Insert" depending on the type of ItemTemplate.
	`<ItemTemplate>`
	`  <desDD:DynamicPattern id="ReadOnlyFields" runat="server"`
	`  PatternTemplateName="DetailsViewButtons" Mode="ReadOnly" >`
	`  </desDD:DynamicPattern>`
	`</ItemTemplate>`

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

Using actual Button controls

You can add actual DES Button controls into a TemplateField object (for GridView and DetailsView controls) or TemplateNotInPattern object (for DynamicPattern, DynamicFormView, and DynamicListView controls). When you do this, you must do most of the setup:

- Assigning the **CommandName** property to the correct command.
- Set **CausesValidation** = `false` to prevent the button from validating.
- Supply the **ValidationGroup** name.
- Hide it if it does not apply to the situation. For example, don't show an Edit button when the DataSource does not support edit mode.
- Assign properties to enhance buttons, like **DisableOnSubmit** and **ChangeMonitorEnabled**, as the Customizer control cannot supply them.

Click on any of these topics to jump to them:

- Buttons for ReadOnly Mode in a list
- Buttons for ReadOnly Mode in single record view
- Buttons for Insert mode
- Buttons for Edit mode
- Applying the Validation Group name and Customizer settings
- Applying rules from the Customizer control using DataBinding

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

Buttons for ReadOnly Mode in a list

There are four possible buttons to offer the user on each row of a list. Here are their command names and actions. Command names are case sensitive.

Button label	CommandName	Action
Edit	"Edit"	Use the list to edit the current row
	"GoToEdit"	Go to a single record view to edit the current row
Delete	"Delete"	Delete the current row. Always setup the **ConfirmMessage** property to avoid letting the user click on this button without confirmation.
Details	"Details"	Go to a single record view in ReadOnly mode for the current row
Select	"Select"	Identify the current row as "selected".

Here is an example showing all buttons, separated by ` `. Add them to a TemplateNotInPattern object (for DynamicPattern, DynamicFormView, and DynamicListView controls) or TemplateField object (for GridView and DetailsView controls).

Note: You can use DES's LinkButton or ImageButton as well.

```
<des:Button ID="EditButton" runat="server" Text="Edit"
   CommandName="Edit" InAJAXUpdate="True" CausesValidation="False" /> 
<des:Button ID="DeleteButton" runat="server" Text="Delete"
   CommandName="Delete" InAJAXUpdate="True" CausesValidation="False"
   ConfirmMessage="Are you sure that you want to delete?" /> 
<des:Button ID="DetailsButton" runat="server" Text="Details"
   CommandName="Details" InAJAXUpdate="True" CausesValidation="False" /> 
<des:Button ID="SelectButton" runat="server" Text="Select"
   CommandName="Select" InAJAXUpdate="True" CausesValidation="False" />
```

Buttons for ReadOnly Mode in single record view

There are three buttons to offer the user on each row of a list. Here are their command names and actions. Command names are case sensitive.

Button label	CommandName	Action
Edit	"Edit"	Edit the current record. The DataBound Control Adapter's **CommandsUseRouting** and **EditCommandUrl** properties determine what page shows the record in edit mode.
Delete	"Delete"	Delete the current row. Always setup the **ConfirmMessage** property to avoid letting the user click on this button without confirmation.
New	"New"	Create a new record. The DataBound Control Adapter's **CommandsUseRouting** and **InsertCommandUrl** properties determine what page shows the record in insert mode.

Here is an example showing all buttons, separated by ` `. Add them to a TemplateNotInPattern object (for DynamicPattern, DynamicFormView, and DynamicListView controls) or TemplateField object (for GridView and DetailsView controls).

Note: You can use DES's LinkButton or ImageButton as well.

```
<des:Button ID="EditButton" runat="server" Text="Edit"
   CommandName="Edit" InAJAXUpdate="True" CausesValidation="False" /> 
<des:Button ID="DeleteButton" runat="server" Text="Delete"
   CommandName="Delete" InAJAXUpdate="True" CausesValidation="False"
   ConfirmMessage="Are you sure that you want to delete?" /> 
<des:Button ID="InsertButton" runat="server" Text="New"
   CommandName="New" InAJAXUpdate="True" CausesValidation="False" />
```

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

Buttons for Insert mode

Insert mode needs two buttons. Here are their command names and actions. Command names are case sensitive.

Button label	CommandName	Action
Save	"Insert"	Save the record.
Cancel	"Cancel"	Do not save the record. Return to another mode or page.

Here is an example showing all buttons, separated by ` `. Add them to a TemplateNotInPattern object (for DynamicPattern, DynamicFormView, and DynamicListView controls) or TemplateField object (for GridView and DetailsView controls).

Note: You can use DES's LinkButton or ImageButton as well.

```
<des:Button ID="SaveButtonInsert" runat="server" Text="Save"
   CommandName="Insert" InAJAXUpdate="True" /> 
<des:Button ID="CancelButtonInsert" runat="server" Text="Cancel"
   CommandName="Cancel" InAJAXUpdate="True" CausesValidation="False" />
```

Buttons for Edit mode

There are four possible buttons to offer the user on each row of a list. Here are their command names and actions. Command names are case sensitive.

Button label	CommandName	Action
Save	"Update"	Save the record.
Cancel	"Cancel"	Do not save the record. Return to another mode or page.

Here is an example showing all buttons, separated by ` `. Add them to a TemplateNotInPattern object (for DynamicPattern, DynamicFormView, and DynamicListView controls) or TemplateField object (for GridView and DetailsView controls).

Note: You can use DES's LinkButton or ImageButton as well.

```
<des:Button ID="SaveButtonEdit" runat="server" Text="Save"
   CommandName="Update" InAJAXUpdate="True" /> 
<des:Button ID="CancelButtonEdit" runat="server" Text="Cancel"
   CommandName="Cancel" InAJAXUpdate="True" CausesValidation="False" />
```

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

Applying the Validation Group name and Customizer settings programmatically

DES can search through a control containing your buttons to setup the Validation Group and apply these properties from the Customizer control: **DisableOnSubmit**, **MayMoveOnClick**, **ChangeMonitorEnables**, **DeleteCommandMessage** (when CommandName="Delete"), **UpdateConfirmMessage** (when CommandName="Insert"), and **CancelConfirmMessage** (when CommandName="Cancel").

In the Page's `OnPreRender()` method, call the `SetupButtonsOnClientSide()` method on the DynamicDataManager control. See "SetupButtonsOnClientSide method".

Applying rules from the Customizer control using DataBinding

Apply rules from the Customizer control, such as **DeleteConfirmMessage** and **DisableOnSubmit**, to related properties on the button using databinding.

If you have a Customizer control within the scope of the control that needs it, you can refer to it directly, like this:

```
<des:Button ID="DeleteButton" runat="server" Text="Delete"
   CausesValidation="false"
   ConfirmMessage="<%# Customizer1.DeleteConfirmMessage %>"
   CommandName="Delete" InAJAXUpdate="True" />
```

Otherwise, let DES find you the right Customizer control by calling the `GetCustomizer()` method on the DynamicDataManager control. See "".

```
<des:Button ID="DeleteButton" runat="server" Text="Delete"
   CausesValidation="false"
   ConfirmMessage="<%#
      DynamicDataManager1.GetCustomizer(FormView1).DisableOnSubmit%>"
   CommandName="Delete" InAJAXUpdate="True" />
```

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

Redirecting to another PageTemplate on a button click

Page Templates are designed to switch between web forms based on the mode. They use the ASP.NET URL Routing system to do this, by changing the URLs. There are several cases to consider:

- Buttons contained within the DataBound control and have their **CommandName** property set to a command recognized by the DataBound control.
- Buttons outside of the DataBound control can use several techniques, including using the `MetaTable.GetActionPath()` method (below), and RunActionForCommand method on the DynamicDataManager control.
- Hyperlinks can be replaced by the DynamicHyperLink control to establish the correct URL. The DynamicHyperLink control comes with the 7/16/08 Futures release and preview versions of ASP.NET 4. If you do not want to use it, see "Redirect using a Hyperlink".

Using the GetActionPath method

When any button posts back to switch modes, it should use the `Response.Redirect()` method and get the desired URL from the MetaTable's `GetActionPath()` method. You can also setup a HyperLink control's URL to use the `GetActionPath()` method. The method takes one or two parameters.

```
GetActionPath(ActionName)
GetActionPath(ActionName, DataItem)
```

ActionName is required. It is a string with one of these values: "List", "Details", "Edit", or "Insert".

DataItem is only required for Details and Edit commands. It provides data about the current record used to build a querystring parameter on the URL. Get its value from the GetDataItem() method found on the Page.

Both return a string containing a URL.

Example: Redirect to Insert.aspx using Button or ImageButton

When you use a Button, LinkButton or ImageButton for your Insert command, it will post back to perform its action. You must implement its **OnClick** event handler to call `Response.Redirect()`. Pass the result of the `MetaTable.GetActionPath(PageAction.Insert)` method.

[C#]

```
protected void Button1_Click(object pSender, EventArgs pE)
{
  Response.Redirect(table.GetActionPath("Edit", GetDataItem());
}
```

[VB]

```
Protected Sub Button1_Click(ByVal pSender As Object, ByVal pE As EventArgs)
  Response.Redirect(table.GetActionPath("Edit", GetDataItem())
End Sub
```

Example: Redirect to Details.aspx or Edit.aspx using Button or ImageButton

When you use a Button, LinkButton or ImageButton for your commands, they will post back to perform their action. While you can handle their postback, its not always easy to get access to the data passed to the second parameter of `MetaTable.GetActionPath()`. Instead, you will setup the client-side onclick DHTML event to redirect.

This example goes to **Edit.aspx**, **Insert.aspx** and **Details.aspx**.

```
<des:Button ID="EditButton" runat="server"
  OnClientClick='<%# "location.href=" +
    table.GetActionPath(PageAction.Edit, GetDataItem()) %>' Text="Edit" /> 
<des:Button ID="InsertButton" runat="server"
  NavigateUrl='<%# "location.href=" + table.GetActionPath(PageAction.Insert) %>'
  Text="New" /> 
```

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

```
<des:Button ID="DetailsButton" runat="server"
   NavigateUrl='<%# "location.href=" +
      table.GetActionPath(PageAction.Details, GetDataItem()) %>' Text="Details" />
```

Redirect using a Hyperlink

Note: Prereleases for ASP.NET 4 include a DynamicHyperLink control. It can be used instead of the HyperLink control to achieve the same functionality as shown here.

When you want to redirect without a postback, use a Hyperlink control in the TemplateField. Set its **NavigateUrl** property to the result of the `MetaTable.GetActionPath()` method. This example goes to **Edit.aspx**, **Insert.aspx** and **Details.aspx**.

```
<asp:HyperLink ID="EditHyperLink" runat="server"
   NavigateUrl='<%# table.GetActionPath(PageAction.Edit, GetDataItem()) %>'
   Text="Edit" /> 
<asp:HyperLink ID="InsertHyperLink" runat="server"
   NavigateUrl='<%# table.GetActionPath(PageAction.Insert) %>'
   Text="New" /> 
<asp:HyperLink ID="DetailsHyperLink" runat="server"
   NavigateUrl='<%# table.GetActionPath(PageAction.Details, GetDataItem()) %>'
   Text="Details" />
```

The first parameter is the enumerated type `System.Web.DynamicData.PageAction`. In addition to `Edit`, `Insert`, and `Details`, it offers `List`.

The second parameter uses the current row of the DataBound control to determine the exact record to display. It is not used with the Insert action, allowing that command to be on a button outside of the DataBound control.

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

Creating Table-Specific PageTemplates

ALERT: DES uses a different architecture for Page Templates. If you have web forms in the ~/DynamicData/CustomPages folder built using native DynamicData, you will need to replace them.

The web forms in the Page Templates folder always use Automatic Scaffolding to determine the list of data fields they show. You can override this behavior in two ways:

Customizing Automatic Scaffolding

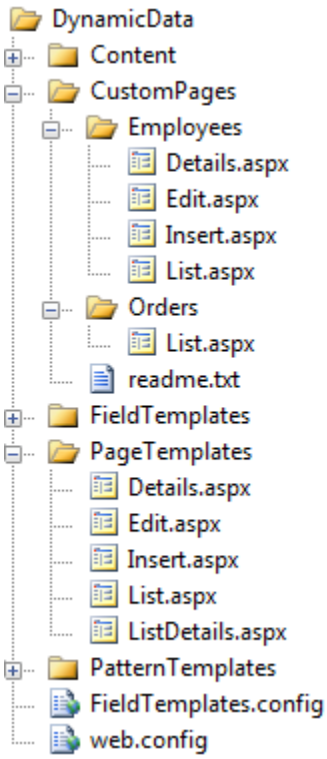
Implement the IPatternTemplateFieldGeneratorGuidance interface to programmatically specify how the scaffolding builds its list. See "Customizing Automatic Scaffolding".

This approach continues to use the files in the **[web application]/DynamicData/PageTemplates** folder which means that the overall user interface is the same. If you have a different user interface in mind, you can either use the CustomPages folder (below), or implement the ICustomizeDataBoundControl interface. See "Implementing the ICustomizeDataBoundControl Interface".

Using the CustomPages folder

The **[web application]/DynamicData/CustomPages** folder declares Page Templates specific to individual tables. You generally populate the DynamicListView or DynamicFormView controls in those web forms with your desired list of data fields in their **Fields** properties. You can also customize the interface, going so far as switching out the DataBound controls with others.

Steps to using the CustomPages folder



1. Determine a table that you want to specify the columns and create a folder with that table's name under the CustomPages folder.
2. Copy one or more web forms from the **[web application]/DynamicData/PageTemplates** folder to the newly created table folder. Any web form that you want to use Automatic Scaffolding and already has the correct format can be omitted.

   In the picture to the right, there Page Templates specific to the Employees and Orders tables. The Orders table only replaces the **List.aspx** file and continues to use the other files in the **PageTemplates** folder.
3. Open each table-specific Page Template web form in an editor.
4. Modify the web forms. Continue with one of these:

- Using DynamicListView and DynamicFormView Controls
- Using ListView and FormView Controls
- Using GridView and DetailsView Controls

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

Using DynamicListView and DynamicFormView Controls

The DynamicListView and DynamicFormView control support easy replacement of Automatic Scaffolding by adding DataFieldInPattern and other ItemInPattern objects to their **Columns** and **Fields** properties.

ItemInPattern Class	Usage
DataFieldInPattern	Adds one Data Field to the list. This uses the standard user interface of the Pattern Template, as declared in the <DataFieldTemplate>, <DataFieldItemTemplate>, and <DataFieldItemTemplate> nodes of the file. It can also use a NamedPart when the name is specified in the **NamedPart** property. See "The DataFieldInPattern Class".
DataFieldListInPattern	Add several Data Fields using the HTML and controls of a <NamedPart> node in the Pattern Template. The NamedPart can host many DynamicControls and DynamicLabels, providing a different interface than the <ItemTemplate> node. Populate its **ItemsInPattern** property with **DataFieldInPattern** objects, one for each DynamicControl. Assign the name of the NamedPart to the **NamedPart** property. See "The DataFieldListInPattern Class".
ScaffoldInPattern	Create a list of Data Fields using Automatic Scaffolding. Each field will have its own row or column in Templates of the file. It can also use a NamedPart when the name is specified in the **NamedPart** property. See "The ScaffoldInPattern Class".
ButtonsInPattern	Uses a NamedPart to provide buttons. It can provide titles to the DynamicColumnTitle and DynamicLabel controls in the header and footer rows through its **HeaderText**, and **FooterText** properties. See "The ButtonsInPattern Class".
LabelInPattern	Uses a NamedPart to assign text to a DynamicLabel control through its **LabelText**, **HeaderText**, and **FooterText** properties. See "The LabelInPattern Class".
NoDataFieldsInPattern	Uses a NamedPart to provide additional formatting that lacks DynamicControls and DynamicLabels. For example, the UI for buttons is specified with this class. See "The NoDataFieldsInPattern Class".
TemplateNotInPattern	Declares HTML and web controls, including DynamicControls and DynamicLabels. However, DynamicControls and DynamicLabels must have their DataField properties expressly assigned. See "The TemplateNotInPattern Class".

CONTINUED ON THE NEXT PAGE

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

If you want to customize the user interface of the DataBound control, generally you should edit a Pattern Template. DynamicListView uses the "GridView" Pattern Template by default. DynamicFormView uses the "DetailsView" Pattern Template by default. Either edit them directly or duplicate them and edit the new files.

1. Add a list of ItemInPattern objects (see the above list) to the **ItemsInPattern** collection. A discussion of each type is shown following the example.
2. If you added a ButtonsInPattern object to show buttons, set **AutoGenerateButtons** to `None`.

Example: DetailsView.aspx for Product table

```
<desDD:DynamicFormView ID="RecordDetails" runat="server"
   DataSourceID="RecordDetailsDataSource"
   AutoGenerateButtons="None" >
   <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="ProductID" LabelText="ID"/>
      <desDD:DataFieldInPattern DataField="ProductName" />
      <desDD:DataFieldInPattern DataField="Price" />
      <desDD:ButtonsInPattern NamedPart="LinkButtons"/>
   </ItemsInPattern>
</desDD:DynamicFormView>
```

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

Using ListView and FormView Controls

These controls allow the most customization capabilities, because you have to declare every HTML element. Dynamic Data is added by using DynamicControl, DynamicLabel, and DynamicPattern controls. Because the DynamicListView and DynamicFormView controls allow similar customization without writing much HTML, consider using them instead of ListView and FormView. See "Using DynamicListView and DynamicFormView Controls".

Click on any of these topics to jump to them:

- DES.DynamicControl: The heart of Dynamic Data
- The DynamicLabel Control
- The DynamicPattern Control

Switching from GridView or DetailsView

Use these steps to switch from GridView to ListView and DetailsView to FormView because you want more control over the HTML output. ListView and FormView will use the DynamicControl class instead of the DynamicField class to let ASP.NET Dynamic Data apply its Field Templates to columns.

Recommendation: Switch to DynamicListView and DynamicFormView instead of ListView and FormView to take advantage of Pattern Templates. They are direct subclasses of ListView and FormView and can either generate the markup for you, or you can define your own.

1. Replace the GridView with a ListView or DynamicListView control. Replace the DetailsView with a FormView or DynamicFormView control.
2. Set its **DataSourceID** property to the appropriate DataSource control on the page.
3. Create your templates. Where you want ASP.NET Dynamic Data to fill in control, add the DynamicControl.

Switching from DynamicListView or DynamicFormView

You don't need to switch from DynamicListView to ListView or DynamicFormView to FormView because as direct subclasses of those controls, you can simply override the built-in generation of HTML by defining your own HTML and controls in their various templates.

For DynamicListView, these templates automatically generate markup until declare your markup in them: ItemTemplate, EditItemTemplate, InsertItemTemplate, SelectedItemTemplate, and AlternatingItemTemplate.

For DynamicFormView, these templates automatically generate markup until declare your markup in them: ItemTemplate, EditItemTemplate, InsertItemTemplate.

Its easy to start your own markup with the markup that these controls would generate automatically.

1. Open the web form in Visual Studio and switch to Design mode.
2. Click on the SmartTag button.
3. Choose the appropriate the **Manage Auto Generation of Templates** command.
4. From the list, change templates from "Auto Generate" to "From Template" or back.

AN EXAMPLE IS ON THE NEXT PAGE

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

Example: FormView

Here is a snippet that can be used in **CustomPages/Products/Details.aspx**. It is a two column table, showing the fields ProductName and UnitPrice. The Edit, Insert, and Delete buttons come from the "DetailsViewButtons" Pattern Template file as declared in the DynamicPattern control.

```
<asp:FormView ID="FormView1" runat="server" DataSourceID="DetailsDataSource" >
   <ItemTemplate>
   <table>
      <tr>
         <td>
            <desDD:DynamicLabel id="DL1" runat="server"
               AssociateControlID="ProductNameReadOnly" />
         </td>
         <td>
            <desDD:DynamicControl ID="ProductNameReadOnly" runat="server"
               DataField="ProductName" />
         </td>
      </tr>
      <tr>
         <td>
            <desDD:DynamicLabel id="DL2" runat="server"
               AssociateControlID="UnitPriceReadOnly" />
         </td>
         <td>
            <desDD:DynamicControl ID="UnitPriceReadOnly" runat="server"
               DataField="UnitPrice" UIHint="Currency" />
         </td>
      </tr>
      <desDD:DynamicPattern ID="ButtonsReadOnly" runat="server"
         PatternTemplateName="DetailsViewButtons" />
   </table>
   </ItemTemplate>
</asp:FormView>
```

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

Using GridView and DetailsView Controls

At this point, you can continue using the GridView and DetailsView controls or replace them with DynamicListView and DynamicFormView controls, because they are more flexible with their presentation. See "Using DynamicListView and DynamicFormView Controls".

1. Change the **AutoGenerateRows** (in DetailsView) or **AutoGenerateColumns** (in GridView) property to `false`.
2. Add a list of DynamicField elements to the **Columns** (GridView) or **Fields** (DetailsView) property. You can use either the `<asp:DynamicField>` or `<desDD:DynamicField>` although DES's control is useful in several situations. See "DES.DynamicField: Using Dynamic Data in GridView and DetailsView".
3. If you want to specify buttons in the list, first set these properties to `false`: **AutoGenerateEditButton**, **AutoGenerateInsertButton** (only in DetailsView), **AutoGenerateDeleteButton**, and **AutoGenerateSelectButton** (only in GridView). Then add either a CommandField or ButtonField.

Example: DetailsView

Here is a snippet that can be used in **CustomPages/Products/Details.aspx**. It is a two column table, showing the fields ProductName and UnitPrice. The Edit, Insert, and Delete buttons come from the "DetailsViewButtons" Pattern Template file as declared in the DynamicPattern control.

```
<asp:DetailsView ID="DetailsView1" runat="server"
  AutoGenerateRows="False" AutoGenerateInsertButton="False"
  AutoGenerateEditButton="False" AutoGenerateDeleteButton="False"
  DataSourceID="DetailsDataSource" DefaultMode="Edit" >
  <Fields>
    <desDD:DynamicField DataField="ProductName" />
    <desDD:DynamicField DataField="UnitPrice" UIHint="Currency" />
    <asp:CommandField ShowEditButton="True" ShowInsertButton="True" />
  </Fields>
</asp:DetailsView>
```

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

Customizing Automatic Scaffolding

Automatic Scaffolding defines the data fields shown tables in DataBound controls so you don't have define them manually. For example, the GridView control will populate its **Columns** property with DynamicFields when you set **AutoGenerateColumns** to `true`. Similarly, the DynamicFormView and DynamicListView controls will populate their **ItemsInPattern** collection with DataFieldInPattern objects when its **ItemsInPattern** collection is empty or you set its **AutoScaffoldDataFields** property to `True`.

Page Templates are setup with this feature enabled, allow them to work with any table. See "Automatic Scaffolding Overview".

The fields are selected and ordered by a **FieldGenerator object**. DES Dynamic Data defines this object for you, replacing one that comes with native Dynamic Data, to allow extensive customization of what is generated and extending the ScaffoldColumnAttribute.

DES's FieldGenerator class will evaluate the properties on the EntityTable Metadata Class. So long as they don't use the ScaffoldColumnAttribute to hide them, they will appear. They appear in the order defined in the EntityTable Class, unless you use ordering properties on the ScaffoldColumnAttribute.

You can programmatically customize the list of fields by implementing either the IPatternTemplateFieldGeneratorGuidance interface (for DynamicListView, DynamicFormView, or DynamicPattern controls) or IDataControlFieldGeneratorGuidance interface (for GridView or DetailsView controls).

Click on any of these topics to jump to them:

- Automatic Scaffolding Overview
- Different Approaches for Customization
- The IPatternTemplateFieldGeneratorGuidance interface
- The IDataControlFieldGeneratorGuidance interface
- PeterBlum.DES.DynamicData.FieldGenerationArgs
  - FieldGenerationArgs Properties
  - AddColumn() method
  - AddField() method
  - GetFieldByColumnName() method
  - HideFieldByColumnName() method
  - MoveFieldByColumnName() method
- Creating your own FieldGenerator Class

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

Different Approaches for Customization

What happens if you want to customize Automatic Scaffolding?

Click on any of these topics to jump to them:

- Customizing with native Dynamic Data
- Customizing with DES Dynamic Data
  - Features of the IFieldGeneratorGuidance interfaces
- Customizing your own way

Customizing with native Dynamic Data

ASP.NET Dynamic Data has you create duplicate Page Template web forms and put them in the **/DynamicData/CustomPages/[tablename]** folder. Then you set **AutoGenerateRows**/**AutoGenerateColumns** to `false` and populate the **GridView.Columns** or **DetailsView.Fields** collections with your DataControlFields (DynamicFields, ButtonFields, TemplateFields, etc). See "Creating Table-Specific PageTemplates".

This methodology has its limitations:

- Each table has a single Page Template representing each action. If you want that columns list to vary based on the situation, the model does not work. Instead you will programmatically populate the **GridView.Columns** or **DetailsView.Fields** collection.
- If you want to change the overall look of your Page Templates, say add a new Label control to all **Edit.aspx** files, you have to carefully apply the changes to all Page Templates. This is prone to errors.

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

Customizing with DES Dynamic Data

DES Dynamic Data uses a more advanced FieldGenerator class, `PeterBlum.DES.DynamicData.DESFieldGenerator`. It lets you dictate how the automatic scaffolding works so you don't have to copy Page Template web forms and maintain them. You continue using the Page Templates in the **/DynamicData/PageTemplates** folder.

Your DataBoundControlAdapter, which is defined in the DynamicDataManager control, hosts several settings to customize the process in its **ScaffoldingSettings** property. They include whether the ScaffoldColumnAttribute is involved and using the CategoryAttribute to determine which columns are included. See "Automatic Scaffolding Properties".

```
<desDD:DynamicDataManager ID="DynamicDataManager1" runat="server"
   AutoLoadForeignKeys="true" >
   <Adapters>
      <desDD:DynamicFormViewAdapter DataBoundControlID="RecordDetails"
         ValidationSummaryControlID="ValidationSummary1"
         SupportsEditMode="true" SupportsInsertMode="false" >
         <ScaffoldingSettings Property="value" />
      </desDD:DynamicFormViewAdapter>
   </Adapters>
</desDD:DynamicDataManager>
```

If the features of **ScaffoldingSettings** and the ScaffoldingColumnAttribute are not flexible enough for you, implement one of these interfaces:

- IPatternTemplateFieldGeneratorGuidance – For DynamicListView, DynamicFormView, and DynamicPattern controls.
- IDataControlFieldGeneratorGuidance – For GridView and DetailsView controls.

Note: You don't create the DESFieldGenerator object. Its done automatically within the DynamicDataManager control.

Features of the IFieldGeneratorGuidance interfaces

- You can define the DataFieldInPattern or DynamicField objects yourself or let the scaffolding rules generate it.
- You can populate the fields list with alternatives to DataFieldInPattern or DynamicField, like TemplateNotInPattern or TemplateField. If you subclass DataFieldInPattern or DynamicField, you can define your subclass.
- After Automatic Scaffolding generates it, you can edit the field objects it created. Remove, reorder, or change properties on the field object.
- Setup field objects based on the DataBound control's usage (List, Detail, Insert, Edit), mode (ReadOnly, Edit, Insert), or style (List oriented vs single record oriented).
- Programmatically determine which columns are included by the automatic scaffolding rules. Effectively you override the ScaffoldColumnAttribute and do it yourself programmatically.
- Override several of the hardcoded rules of the native FieldsGenerator.

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

Customizing your own way

You can also create a traditional FieldGenerator class and assign it to your DataBound Control Adapter's FieldGenerator property. See "Creating your own FieldGenerator Class".

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

The IPatternTemplateFieldGeneratorGuidance interface

There are two interfaces for customizing the DESFieldGenerator's behavior.

Interface	Used with
IPatternTemplateFieldGeneratorGuidance	Pattern Templates. In DynamicPattern, DynamicListView, and DynamicListView controls. Described below.
IDataControlFieldGeneratorGuidance	GridView and DetailsView controls. See "The IDataControlFieldGeneratorGuidance interface".

The DESFieldGenerator class behaves just like the native FieldGenerator class until you implement the `PeterBlum.DES.DynamicData.IPatternTemplateFieldGeneratorGuidance` interface. This interface has three methods as shown here:

[C#]

```
public interface IPatternTemplateFieldGeneratorGuidance
{
   void BeforeGeneration(
     FieldGenerationArgs<DataFieldInPattern, BaseItemInPattern> pArgs);

   DataControlField CreateField(MetaColumn pColumn,
     FieldGenerationArgs<DataFieldInPattern, BaseItemInPattern> pArgs);

   void AfterGeneration(
     FieldGenerationArgs<DataFieldInPattern, BaseItemInPattern> pArgs);
}
```

[VB]

```
Public Interface IPatternTemplateFieldGeneratorGuidance
  Sub BeforeGeneration(ByVal pArgs As FieldGenerationArgs _
    (Of DataFieldInPattern, BaseItemInPattern))

  Function CreateField(ByVal pColumn As MetaColumn, _
    ByVal pArgs As FieldGenerationArgs _
    (Of DataFieldInPattern, BaseItemInPattern)) As DataControlField

  Sub AfterGeneration(ByVal pArgs As FieldGenerationArgs _
    (Of DataFieldInPattern, BaseItemInPattern))
End Interface
```

The implementation of each method is shown in the next topics. All of them utilize the *pArgs* parameter. You will properties on *pArgs* to guide the DESFieldGenerator. See "PeterBlum.DES.DynamicData.FieldGenerationArgs".

The main idea is to populate the **pArgs.Fields** collection, which is the list of field objects to show in the DataBound control. You can either populate it yourself by using the `BeforeGeneration()` method, or let the DESFieldGenerator populate it. When the DESFieldGenerator needs to create a field object, it calls the `CreateField()` method where you can define the object you want. Use the `AfterGeneration()` method to customize the list after its generated.

By default, *pArgs* is setup to run automatic field generation using scaffolding.

Click on any of these topics to jump to them:

- Where to implement IPatternTemplateFieldGeneratorGuidance
- Adding the IPatternTemplateFieldGeneratorGuidance Interface
- IPatternTemplateFieldGeneratorGuidance.BeforeGeneration method
- IPatternTemplateFieldGeneratorGuidance.CreateField method
- IPatternTemplateFieldGeneratorGuidance.AfterGeneration method

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

Where to implement IPatternTemplateFieldGeneratorGuidance

As an interface, you attach it to a class then create the list of methods within that class. You already have some classes created where you should consider attaching it: the Page, the MasterPage, or any UserControl containing the DynamicDataManager control. If you use any of these, DES will find and use it. You can also explicitly assign the class to the Customizer control's **PatternTemplateFieldGeneratorGuidanceOwner** property.

How DES locates your interface

You may have this interface declared in a few classes, such as both the MasterPage and UserControl containing a DynamicDataManager. If you let DES search for the interface, it always searches in this order:

1. User control with DynamicDataManager
2. Master Page
3. Page

How to specify which object to use

If you need to specify which of these to use, assign the object to the Customizer control's **PatternTemplateFieldGeneratorGuidanceOwner** property.

Always set the **PatternTemplateFieldGeneratorGuidanceOwner** property in the DynamicDataManager's Init event handler method or `Page_Init()` method.

[C#]

```
Customizer1.PatternTemplateFieldGeneratorGuidanceOwner = this;
```

[VB]

```
Customizer1.PatternTemplateFieldGeneratorGuidanceOwner = Me
```

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

Adding the IPatternTemplateFieldGeneratorGuidance Interface

You can add this interface to the Master Page, Page, or a User Control containing the Dynamic Data controls.

1. Start by using the `PeterBlum.DES.DynamicData` namespace. It's easiest to map it to the name "DES" just like was done in the code file containing the EntityTable Metadata Classes.

   [C#]

   ```
   using DES = PeterBlum.DES.DynamicData;
   ```

   [VB]

   ```
   Import DES = PeterBlum.DES.DynamicData
   ```

2. Assign the IPatternTemplateFieldGeneratorGuidance interface to the desired class.

   [C#]

   ```
   public class WebForm1 : System.Web.UI.WebControls.Page,
       DES.IPatternTemplateFieldGeneratorGuidance
   ```

   [VB]

   ```
   Public Class WebForm1
      Implements System.Web.UI.WebControls.Page, _
      Implements DES.IPatternTemplateFieldGeneratorGuidance
   ```

3. Implement the `BeforeGeneration()` method. Use it to either determine how automatic field generation works or to populate the **pArgs.Fields** collection. See "IPatternTemplateFieldGeneratorGuidance.BeforeGeneration method".

4. Implement the `CreateField()` method. Use it to create the DataFieldInPattern object associated with the column or to prevent that column from being added. See "IPatternTemplateFieldGeneratorGuidance.CreateField method".

5. Implement the `AfterGeneration()` method. Use it to modify the results of automatic field generation. See "IPatternTemplateFieldGeneratorGuidance.AfterGeneration method".

6. If its not already on the page, add the Customizer control. It is the conduit between your method and the code that invokes it. When using Page Templates, typically the Customizer is added to the MasterPage.

7. DES Dynamic Data will discover your control or page class that implements IPatternTemplateFieldGeneratorGuidance so long as that class is a User Control that contains the DES DynamicDataManager control, the MasterPage, or the Page. See "How DES locates your interface".

   If it cannot be discovered, add a Customizer control and assign the object in either the DynamicDataManager's Init event handler or the `Page_Init()` method like this:

   [C#]

   ```
   Customizer1.PatternTemplateFieldGeneratorGuidanceOwner = this;
   ```

   [VB]

   ```
   Customizer1.PatternTemplateFieldGeneratorGuidanceOwner = Me
   ```

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

IPatternTemplateFieldGeneratorGuidance.BeforeGeneration method

Called before automatic field generation to let you do several things:

- Populate the **pArgs.Fields** collection. In doing so, you turn off automatic field generation unless you also set **pArgs.UseAutoGenerate** is set to `UseAutoGenerate.Yes`. pArgs has methods to easily add columns by name (`AddColumn()`) or by DataFieldInPattern object (`AddField()`).
- Determine the rules for automatic field generation with these properties on the pArgs parameter: **UseScaffoldColumnAttribute**, **Categories**, **ShowLongStringsInLists**, **ShowReadOnlyColumnsInEditMode**. and **ShowChildrenTemplateInEditMode**. See "FieldGenerationArgs Properties"

If you don't need to take these actions, still define `BeforeGeneration()`, leaving it an empty method. Otherwise modify the pArgs parameter.

- Example: Populate Fields
- Example: Insert button controls and use Automatic Fields Generation
- Example: Use Automatic Fields Generation with default settings
- Example: Use Automatic Fields Generation with custom settings

Example: Populate Fields

This will create populate lists with just three columns on the Employees table: FirstName, LastName, BirthDate. The LastName uses `AddField()` merely as an example for that function. Otherwise, it is the same as calling `AddColumn("LastName")`.

All other tables and usages will use automatic field generation using scaffolding because pArgs is defaults to work that way.

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public void BeforeGeneration(
   DES.FieldGenerationArgs<DES.DataFieldInPattern, DES.BaseItemInPattern> pArgs)
{
   switch (pArgs.TableName)
   {
      case "Employees":
         if (pArgs.DataControlAction == DES.DataControlAction.List)
         {
            pArgs.AddColumn("FirstName");
            pArgs.AddField(
               new PeterBlum.DES.DynamicData.DataFieldInPattern("LastName"));
            pArgs.AddColumn("BirthDate");
         }
         break;
   default:
      break;
    } // switch
}
```

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

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Sub BeforeGeneration(ByVal pArgs As _
  DES.FieldGenerationArgs(Of DES.DataFieldInPattern, DES.BaseItemInPattern)) _
    Implements DES.IPatternTemplateFieldGeneratorGuidance.BeforeGeneration

  Select Case pArgs.TableName
    Case "Employees"
      If pArgs.DataControlAction = DES.DataControlAction.List Then
        pArgs.AddColumn("FirstName")
        pArgs.AddField( _
          New PeterBlum.DES.DynamicData.DataFieldInPattern("LastName"))
        pArgs.AddColumn("BirthDate")
      End If
   End Select
End Sub
```

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

Example: Insert button controls and use Automatic Fields Generation

When using the DynamicListView or DynamicFormView, you can insert buttons from Pattern Templates by using the DataFieldListInPattern object.

This code adds a DataFieldListInPattern object for LinkButtons and has automatic scaffolding insert the data fields normally. It limits the work to the "Employees" table. All other tables and usages will use automatic field generation using scaffolding because pArgs is defaults to work that way.

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public void BeforeGeneration(
   DES.FieldGenerationArgs<DES.DataFieldInPattern, DES.BaseItemInPattern> pArgs)
{
   switch (pArgs.TableName)
   {
      case "Employees":
         if (pArgs.DataControlAction == DES.DataControlAction.List)
         {
            DES.DataFieldListInPattern vNIT =
               new DES.DataFieldListInPattern("LinkButton");
            pArgs.AddField(vNIT);
            pArgs.UseAutoGenerate = DES.UseAutoGenerate.Yes;
         }
         break;
   default:
      break;
    } // switch
}
```

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Sub BeforeGeneration(ByVal pArgs As _
   DES.FieldGenerationArgs(Of DES.DataFieldInPattern, DES.BaseItemInPattern)) _
      Implements DES.IPatternTemplateFieldGeneratorGuidance.BeforeGeneration

   Select Case pArgs.TableName
      Case "Employees"
         If pArgs.DataControlAction = _
               DES.DataControlAction.List Then
            Dim vNIT As DES.DataFieldListInPattern = _
               New DES.DataFieldListInPattern("LinkButton")
            pArgs.AddField(vNIT)
            pArgs.UseAutoGenerate = DES.UseAutoGenerate.Yes
         End If
    End Select Case
End Sub
```

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

Example: Use Automatic Fields Generation with default settings

Since pArgs is already setup for this, just declare the empty method.

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public void BeforeGeneration(
   DES.FieldGenerationArgs<DES.DataFieldInPattern, DES.BaseItemInPattern> pArgs)
{
}
```

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Sub BeforeGeneration(ByVal pArgs As _
   DES.FieldGenerationArgs(Of DES.DataFieldInPattern, DES.BaseItemInPattern)) _
      Implements DES.IPatternTemplateFieldGeneratorGuidance.BeforeGeneration

End Sub
```

Example: Use Automatic Fields Generation with custom settings

Use the option to show "long strings" in lists. A long string is a column that doesn't have a maximum length. Your own code might test pArgs.TableName to assign these rules depending on the table.

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public void BeforeGeneration(
   DES.FieldGenerationArgs<DES.DataFieldInPattern, DES.BaseItemInPattern> pArgs)
{
   pArgs.ShowLongStringsInLists = true;
}
```

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Sub BeforeGeneration(ByVal pArgs As _
   DES.FieldGenerationArgs(Of DES.DataFieldInPattern, DES.BaseItemInPattern)) _
      Implements DES.IPatternTemplateFieldGeneratorGuidance.BeforeGeneration

   pArgs.ShowLongStringsInLists = True
End Sub
```

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

IPatternTemplateFieldGeneratorGuidance.CreateField method

Automatic field generation will evaluate all columns in the current table's MetaTable. Based on rules established on pArgs (`PeterBlum.DES.DynamicData.FieldGenerationArgs`), it will add them to **pArgs.Fields**. When it finally determines that a column should be included, it calls `CreateField()` to create the actual object that goes into pArgs.Fields. Usually you just create a `PeterBlum.DES.DynamicData.DataFieldInPattern` with the data field name requested and return it. You can also do this:

- Create the DataFieldInPattern object and set various properties, including those used to set properties on the Field Template that will be created.
- Create a different ItemInPattern class, such as DataFieldInPattern or DataFieldListInPattern, to better define the user interface for this column. A TemplateNotInPattern for instance can contain multiple DynamicControls from your table to show in the same column position.
- Determine if you really want that column. If so, create the DataFieldInPattern. If not, return `null`/`Nothing`. This gives you the final decision on whether the column is shown.

Keep in mind that automatic field generation will populate **pArgs.Fields** in the order determined by the MetaTable.Columns collection and ScaffoldColumnAttributes. If you want to modify the order, use the `AfterGeneration()` method.

Example: The default CreateField method

When using auto field generation, it should always create and return a DataFieldInPattern object for the column it is passed if you have no other rules to apply.

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public DES.BaseItemInPattern CreateField(MetaColumn pColumn,
   FieldGenerationArgs<DES.DataFieldInPattern, DES.BaseItemInPattern> pArgs)
{
   return new DES.DataFieldInPattern(pColumn.Name);
}
```

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Function CreateField(ByVal pColumn As MetaColumn, ByVal pArgs As _
     FieldGenerationArgs(Of DES.DataFieldInPattern, DES.BaseItemInPattern)) _
   As DES.DataFieldInPattern _
   Implements DES.IPatternTemplateFieldGeneratorGuidance.CreateField

   Return New DES.DataFieldInPattern(pColumn.Name)
End Function
```

MORE SAMPLES ON THE NEXT PAGE

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

Example: Customizing a DynamicField

After creating a DataFieldInPattern object, use this method to assign its properties based on the column and the usage. In this case, when on table "Employee" and column "LastName", set the column width using the **DataColumnWidth** property that exists on DES's Field Templates.

You can also take the same action in the `AfterGeneration()` method if you like.

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public DES.BaseItemInPattern CreateField(MetaColumn pColumn,
   FieldGenerationArgs<DES.DataFieldInPattern, DES.BaseItemInPattern> pArgs)
{
   DES.DataFieldInPattern vDF = new DES.DataFieldInPattern(pColumn.Name);
   switch (pArgs.TableName)
   {
      case "Employees":
         switch (pArgs.ColumnName)
         {
            case "LastName":
               vDF.EditPropertyList = "DataControlWidth='200px'";
               break;
            default:
               break;
         } // switch
         break;
   default:
      break;
   } // switch

   return vDF;
}
```

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Function CreateField(ByVal pColumn As MetaColumn, ByVal pArgs As _
      FieldGenerationArgs(Of DES.DataFieldInPattern, DES.BaseItemInPattern)) _
   As DES.DataFieldInPattern _
   Implements DES.IPatternTemplateFieldGeneratorGuidance.CreateField

   Dim vDF As DES.DataFieldInPattern = New DES.DataFieldInPattern(pColumn.Name)
   Select Case pArgs.TableName
      Case "Employees"

         Select Case pColumn.Name
            Case "LastName"
               vDF.EditPropertyList = "DataControlWidth='200px'"
         End Select Case

   End Select Case

   Return vDF
End Function
```

MORE SAMPLES ON THE NEXT PAGE

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

Example: Using CreateField to override the scaffolding

The automatic field generation includes columns that do not specify `ScaffoldColumn(false)` on the EntityTable Metadata class. You can use `CreateField()` to omit columns instead.

First in `BeforeGeneration()`, set **pArgs.UseScaffoldColumnAttribute** to `false`. Then use `CreateField()` to create a DynamicField on the desired columns.

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public void BeforeGeneration(
   DES.FieldGenerationArgs<DES.DataFieldInPattern, DES.BaseItemInPattern> pArgs)
{
   pArgs.UseScaffoldColumnAttribute = false;
}

public DES.DataFieldInPattern CreateField(MetaColumn pColumn,
   FieldGenerationArgs<DES.DataFieldInPattern, DES.BaseItemInPattern> pArgs)
{
   switch (pArgs.TableName)
   {
      case "Employees":
         switch (pArgs.ColumnName)
         {
            case "PhotoPath":
            case "Extension":
            case "Orders":
            case "Employees":
               return null;      // skip these
            default:
               break;
         } // switch
         break;
   default:
      break;
   } // switch

   return new DES.DataFieldInPattern(pColumn.Name);
}
```

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

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Sub BeforeGeneration(ByVal pArgs As _
  DES.FieldGenerationArgs(Of DES.DataFieldInPattern, DES.BaseItemInPattern)) _
    Implements DES.IPatternTemplateFieldGeneratorGuidance.BeforeGeneration

  pArgs.UseScaffoldColumnAttribute = False
End Sub

Public Function CreateField(ByVal pColumn As MetaColumn, ByVal pArgs As _
    FieldGenerationArgs(Of DES.DataFieldInPattern, DES.BaseItemInPattern)) _
  As DES.DataFieldInPattern _
  Implements DES.IPatternTemplateFieldGeneratorGuidance.CreateField

  Select Case pArgs.TableName
    Case "Employees"
      Select Case pColumn.Name
        Case "PhotoPath", "Extension", "Orders", "Employees"
          Return Nothing    ' skip these
      End Select Case
  End Select Case

  Return New DES.DataFieldInPattern(pColumn.Name)
End Function
```

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

IPatternTemplateFieldGeneratorGuidance.AfterGeneration method

Called after automatic field generation to let you modify the **pArgs.Fields** collection. *It is not called if automatic field generation is disabled.* You will take advantage of several methods on pArgs to assist you.

- Remove a column by calling `pArgs.HideFieldByColumnName()`.
- Add a column by calling `pArgs.AddColumn()` or `AddField()`.
- Edit a column's properties by retrieving it with `pArgs.GetFieldByColumnName()`.
- Change the order of a column by calling `pArgs.MoveFieldByColumnName()`. There are two versions. One takes the column name after which the column is moved. The other takes a numeric position for the column.

If you don't need to take these actions, still define `AfterGeneration()`, leaving it an empty method. Otherwise modify the pArgs parameter.

Example: Modify pArgs.Fields

For the Employees table, when showing the Details usage, hide these columns: PhotoPath, Employees, and Orders.

For the Employees table, when in Edit or Insert mode, make the PhotoPath's textbox wider.

For the Customers table, rearrange the columns.

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public void AfterGeneration(
   DES.FieldGenerationArgs<DES.DataFieldInPattern, DES.BaseItemInPattern> pArgs)
{
   switch (pArgs.TableName)
   {
      case "Employees":
         if (pArgs.DataControlAction == DES.DataControlAction.Details)
            {
               pArgs.HideFieldByColumnName("PhotoPath");
               pArgs.HideFieldByColumnName("Employees");
               pArgs.HideFieldByColumnName("Orders");
            }
            else if (pArgs.DataBoundControlMode != DataBoundControlMode.ReadOnly)
            {
               DES.DataFieldInPattern vDF =
                  pArgs.GetFieldByColumnName("PhotoPath") as DES.DataFieldInPattern;
               vDF.EditPropertyList = "DataControlWidth='300px'";
               vDF.InsertPropertyList = "DataControlWidth='300px'";
            }
            break;

      case "Customers":
         pArgs.MoveFieldByColumnName("CompanyName", "ContactName");
         pArgs.MoveFieldByColumnName("ContactTitle", "ContactName");
         pArgs.MoveFieldByColumnName("Country", 999, true); // last
         pArgs.MoveFieldByColumnName("Phone", 1, false); // move relative by 1
         pArgs.MoveFieldByColumnName("CustomerID", 4, true); // column 5
         break;

      default:
         break;
   } // switch
}
```

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

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Sub AfterGeneration(ByVal pArgs As _
  DES.FieldGenerationArgs(Of DES.DataFieldInPattern, DES.BaseItemInPattern)) _
    Implements DES.IPatternTemplateFieldGeneratorGuidance.BeforeGeneration

  Select Case pArgs.TableName
    Case "Employees"
      If pArgs.DataControlAction = DES.DataControlAction.Details Then
        pArgs.HideFieldByColumnName("PhotoPath")
        pArgs.HideFieldByColumnName("Employees")
        pArgs.HideFieldByColumnName("Orders")
      Else If pArgs.DataBoundControlMode <> DataBoundControlMode.ReadOnly Then
        Dim vDF As DES.DataFieldInPattern = _
          CType(pArgs.GetFieldByColumnName("PhotoPath"), DES.DataFieldInPattern)
        vDF.EditPropertyList = "DataControlWidth='300px'"
        vDF.InsertPropertyList = "DataControlWidth='300px'"
      End If

    case "Customers"
      pArgs.MoveFieldByColumnName("CompanyName", "ContactName")
      pArgs.MoveFieldByColumnName("ContactTitle", "ContactName")
      pArgs.MoveFieldByColumnName("Country", 999, True) ' last
      pArgs.MoveFieldByColumnName("Phone", 1, False) ' move relative by 1
      pArgs.MoveFieldByColumnName("CustomerID", 4, True) ' column 5
   End Select Case
End Sub
```

Example: Do nothing with AfterGeneration

Just declare the empty method.

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public void AfterGeneration(
  DES.FieldGenerationArgs<DES.DataFieldInPattern, DES.BaseItemInPattern> pArgs)
{
}
```

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Sub AfterGeneration(ByVal pArgs As _
  DES.FieldGenerationArgs(Of DES.DataFieldInPattern, DES.BaseItemInPattern)) _
    Implements DES.IPatternTemplateFieldGeneratorGuidance.BeforeGeneration

End Sub
```

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

The IDataControlFieldGeneratorGuidance interface

There are two interfaces for customizing the DESFieldGenerator's behavior.

Interface	Used with
IDataControlFieldGeneratorGuidance	GridView and DetailsView controls. Described below.
IPatternTemplateFieldGeneratorGuidance	Pattern Template. In DynamicPattern, DynamicListView, and DynamicListView controls. See "The IPatternTemplateFieldGeneratorGuidance interface".

The DESFieldGenerator class behaves just like the native FieldGenerator class until you implement the `PeterBlum.DES.DynamicData.IDataControlFieldGeneratorGuidance` interface. This interface has three methods as shown here:

[C#]

```
public interface IDataControlFieldGeneratorGuidance
{
   void BeforeGeneration(
     DES.FieldGenerationArgs<DES.DynamicField, DataControlField> pArgs);

   DataControlField CreateField(MetaColumn pColumn,
     DES.FieldGenerationArgs<DES.DynamicField, DataControlField> pArgs);

   void AfterGeneration(
     DES.FieldGenerationArgs<DES.DynamicField, DataControlField> pArgs);
}
```

[VB]

```
Public Interface IDataControlFieldGeneratorGuidance
  Sub BeforeGeneration(ByVal pArgs As DES.FieldGenerationArgs _
    (Of DES.DynamicField, DataControlField))

  Function CreateField(ByVal pColumn As MetaColumn, _
    ByVal pArgs As FieldGenerationArgs _
    (Of DES.DynamicField, DataControlField)) As DataControlField

  Sub AfterGeneration(ByVal pArgs As FieldGenerationArgs _
    (Of DES.DynamicField, DataControlField))
End Interface
```

The implementation of each method is shown in the next topics. All of them utilize the *pArgs* parameter. You will properties on *pArgs* to guide the DESFieldGenerator. See "PeterBlum.DES.DynamicData.FieldGenerationArgs".

The main idea is to populate the **pArgs.Fields** collection, which is the list of field objects to show in the DataBound control. You can either populate it yourself by using the `BeforeGeneration()` method, or let the DESFieldGenerator populate it. When the DESFieldGenerator needs to create a field object, it calls the `CreateField()` method where you can define the object you want. Use the `AfterGeneration()` method to customize the list after its generated.

By default, *pArgs* is setup to run automatic field generation using scaffolding.

Click on any of these topics to jump to them:

- Where to implement IDataControlFieldGeneratorGuidance
- Adding the IDataControlFieldGeneratorGuidance Interface
- IDataControlFieldGeneratorGuidance.BeforeGeneration method
- IDataControlFieldGeneratorGuidance.CreateField method
- IDataControlFieldGeneratorGuidance.AfterGeneration method

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

Where to implement IDataControlFieldGeneratorGuidance

As an interface, you attach it to a class then create the list of methods within that class. You already have some classes created where you should consider attaching it: the Page, the MasterPage, or any UserControl containing the DynamicDataManager control. If you use any of these, DES will find and use it. You can also explicitly assign the class to the Customizer control's **DataControlFieldGeneratorGuidanceOwner** property.

How DES locates your interface

You may have this interface declared in a few classes, such as both the MasterPage and UserControl containing a DynamicDataManager. If you let DES search for the interface, it always searches in this order:

1. User control with DynamicDataManager
2. Master Page
3. Page

How to specify which object to use

If you need to specify which of these to use, assign the object to the Customizer control's **DataControlFieldGeneratorGuidanceOwner** property.

Always set the **DataControlFieldGeneratorGuidanceOwner** property in the DynamicDataManager's Init event handler method or `Page_Init()` method.

[C#]

```
Customizer1.DataControlFieldGeneratorGuidanceOwner = this;
```

[VB]

```
Customizer1.DataControlFieldGeneratorGuidanceOwner = Me
```

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

Adding the IDataControlFieldGeneratorGuidance Interface

You can add this interface to the Master Page, Page, or a User Control containing the Dynamic Data controls.

1. Start by using the `PeterBlum.DES.DynamicData` namespace. It's easiest to map it to the name "DES" just like was done in the code file containing the EntityTable Metadata Classes.

   [C#]

   ```
   using DES = PeterBlum.DES.DynamicData;
   ```

   [VB]

   ```
   Import DES = PeterBlum.DES.DynamicData
   ```

2. Assign the IDataControlFieldGeneratorGuidance interface to the desired class.

   [C#]

   ```
   public class WebForm1 : System.Web.UI.WebControls.Page,
       DES.IDataControlFieldGeneratorGuidance
   ```

   [VB]

   ```
   Public Class WebForm1
      Implements System.Web.UI.WebControls.Page, _
      Implements DES.IDataControlFieldGeneratorGuidance
   ```

3. Implement the `BeforeGeneration()` method. Use it to either determine how automatic field generation works or to populate the **pArgs.Fields** collection. See "IDataControlFieldGeneratorGuidance.BeforeGeneration method".

4. Implement the `CreateField()` method. Use it to create the DynamicField instance associated with the column or to prevent that column from being added. See "IDataControlFieldGeneratorGuidance.CreateField method".

5. Implement the `AfterGeneration()` method. Use it to modify the results of automatic field generation. See "IDataControlFieldGeneratorGuidance.AfterGeneration method".

6. If its not already on the page, add the Customizer control. It is the conduit between your method and the code that invokes it. When using Page Templates, typically the Customizer is added to the MasterPage.

7. DES Dynamic Data will discover your control or page class that implements IDataControlFieldGeneratorGuidance so long as that class is a User Control that contains the DES DynamicDataManager control, the MasterPage, or the Page. See "How DES locates your interface".

   If it cannot be discovered, add a Customizer control and assign the object in either the DynamicDataManager's Init event handler or the `Page_Init()` method like this:

   [C#]

   ```
   Customizer1.DataControlFieldGeneratorGuidanceOwner = this;
   ```

   [VB]

   ```
   Customizer1.DataControlFieldGeneratorGuidanceOwner = Me
   ```

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

IDataControlFieldGeneratorGuidance.BeforeGeneration method

Called before automatic field generation to let you do several things:

- Populate the **pArgs.Fields** collection. In doing so, you turn off automatic field generation unless you also set **pArgs.UseAutoGenerate** is set to `UseAutoGenerate.Yes`. pArgs has methods to easily add columns by name (`AddColumn()`) or by DynamicField instance (`AddField()`).
- Determine the rules for automatic field generation with these properties on the pArgs parameter: **UseScaffoldColumnAttribute**, **ShowLongStringsInLists**, **ShowReadOnlyColumnsInEditMode**, and **ShowChildrenTemplateInEditMode**. See "FieldGenerationArgs Properties"

If you don't need to take these actions, still define `BeforeGeneration()`, leaving it an empty method. Otherwise modify the pArgs parameter.

Example: Populate Fields

This will create populate lists with just three columns on the Employees table: FirstName, LastName, BirthDate. The LastName uses `AddField()` merely as an example for that function. Otherwise, it is the same as calling `AddColumn("LastName")`.

All other tables and usages will use automatic field generation using scaffolding because pArgs is defaults to work that way.

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public void BeforeGeneration(
   DES.FieldGenerationArgs<DES.DynamicField, DataControlField> pArgs)
{
   switch (pArgs.TableName)
   {
      case "Employees":
         if (pArgs.DataControlAction == DES.DataControlAction.List)
         {
            pArgs.AddColumn("FirstName");
            pArgs.AddField(
               new PeterBlum.DES.DynamicData.DynamicField("LastName"));
            pArgs.AddColumn("BirthDate");
         }
         break;
   default:
      break;
    } // switch
}
```

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

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Sub BeforeGeneration(ByVal pArgs As _
  DES.FieldGenerationArgs(Of DES.DynamicField, DataControlField)) _
    Implements DES.IDataControlFieldGeneratorGuidance.BeforeGeneration

  Select Case pArgs.TableName
    Case "Employees"
      If pArgs.DataControlAction = DES.DataControlAction.List Then
        pArgs.AddColumn("FirstName")
        pArgs.AddField( _
          New PeterBlum.DES.DynamicData.DynamicField("LastName"))
        pArgs.AddColumn("BirthDate")
      End If
   End Select
End Sub
```

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

Example: Insert button controls and use Automatic Fields Generation

When using the GridView or DetailsView, you can insert buttons by using the CommandField object.

This code adds a CommandField for LinkButtons and has automatic scaffolding insert the data fields normally. It limits the work to the "Employees" table. All other tables and usages will use automatic field generation using scaffolding because pArgs is defaults to work that way.

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public void BeforeGeneration(
   DES.FieldGenerationArgs<DES.DynamicField, DataControlField> pArgs)
{
   switch (pArgs.TableName)
   {
      case "Employees":
         if (pArgs.DataControlAction == DES.DataControlAction.List)
         {
            CommandField vBF = new CommandField();
            vBF.ButtonType = ButtonType.Link;
            vBF.ShowEditButton = true;
            vBF.ShowInsertButton = true;
            pArgs.AddField(vBF);
            pArgs.UseAutoGenerate = DES.UseAutoGenerate.Yes;
         }
         break;
   default:
      break;
    } // switch
}
```

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Sub BeforeGeneration(ByVal pArgs As _
   DES.FieldGenerationArgs(Of DES.DynamicField, DataControlField)) _
      Implements DES.IDataControlFieldGeneratorGuidance.BeforeGeneration

   Select Case pArgs.TableName
      Case "Employees"
         If pArgs.DataControlAction = DES.DataControlAction.List Then
            Dim vBF As CommandField = New CommandField()
            vBF.ButtonType = ButtonType.Link
            vBF.ShowEditButton = True
            vBF.ShowInsertButton = True
            pArgs.AddField(vBF)
            pArgs.UseAutoGenerate = DES.UseAutoGenerate.Yes
         End If
    End Select Case
End Sub
```

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

Example: Use Automatic Fields Generation with default settings

Since pArgs is already setup for this, just declare the empty method.

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public void BeforeGeneration(
   DES.FieldGenerationArgs<DES.DynamicField, DataControlField> pArgs)
{
}
```

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Sub BeforeGeneration(ByVal pArgs As _
   DES.FieldGenerationArgs(Of DES.DynamicField, DataControlField)) _
      Implements DES.IDataControlFieldGeneratorGuidance.BeforeGeneration

End Sub
```

Example: Use Automatic Fields Generation with custom settings

Use the option to show "long strings" in lists. A long string is a column that doesn't have a maximum length. Your own code might test pArgs.TableName to assign these rules depending on the table.

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public void BeforeGeneration(
   DES.FieldGenerationArgs<DES.DynamicField, DataControlField> pArgs)
{
   pArgs.ShowLongStringsInLists = true;
}
```

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Sub BeforeGeneration(ByVal pArgs As _
   DES.FieldGenerationArgs(Of DES.DynamicField, DataControlField)) _
      Implements DES.IDataControlFieldGeneratorGuidance.BeforeGeneration

   pArgs.ShowLongStringsInLists = True
End Sub
```

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

IDataControlFieldGeneratorGuidance.CreateField method

Automatic field generation will evaluate all columns in the current table's MetaTable. Based on rules established on pArgs (`PeterBlum.DES.DynamicData.FieldGenerationArgs`), it will add them to **pArgs.Fields**. When it finally determines that a column should be included, it calls `CreateField()` to create the actual object that goes into pArgs.Fields. Usually you just create a `PeterBlum.DES.DynamicData.DynamicField` with the data field name requested and return it. You can also do this:

- Create the DynamicField object and set various properties, including those used to set properties on the Field Template that will be created.
- Create a different class of DataControlField, such as TemplateField. A TemplateNotInPattern for instance can contain multiple DynamicControls from your table to show in the same column position.
- Determine if you really want that column. If so, create the DynamicField. If not, return `null`/`Nothing`. This gives you the final decision on whether the column is shown.

Keep in mind that automatic field generation will populate **pArgs.Fields** in the order determined by the MetaTable.Columns collection and ScaffoldColumnAttributes. If you want to modify the order, use the `AfterGeneration()` method.

Example: The default CreateField method

When using auto field generation, it should always create and return a DynamicField object for the column it is passed if you have no other rules to apply.

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public DataControlField CreateField(MetaColumn pColumn,
   FieldGenerationArgs<DES.DynamicField, DataControlField> pArgs)
{
   return new DES.DynamicField(pColumn.Name);
}
```

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Function CreateField(ByVal pColumn As MetaColumn, ByVal pArgs As _
     FieldGenerationArgs(Of DES.DynamicField, DataControlField)) _
   As DES.DynamicField _
   Implements DES.IDataControlFieldGeneratorGuidance.CreateField

   Return New DES.DynamicField(pColumn.Name)
End Function
```

MORE SAMPLES ON THE NEXT PAGE

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

Example: Customizing a DynamicField

After creating a DynamicField object, use this method to assign its properties based on the column and the usage. In this case, when on table "Employee" and column "LastName", set the column width using the **DataColumnWidth** property that exists on DES's Field Templates.

You can also take the same action in the `AfterGeneration()` method if you like.

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public DataControlField CreateField(MetaColumn pColumn,
   FieldGenerationArgs<DES.DynamicField, DataControlField> pArgs)
{
   DES.DynamicField vDF = new DES.DynamicField(pColumn.Name);
   switch (pArgs.TableName)
   {
      case "Employees":
         switch (pArgs.ColumnName)
         {
            case "LastName":
               vDF.EditPropertyList = "DataControlWidth='200px'";
               break;
            default:
               break;
         } // switch
         break;
   default:
      break;
   } // switch

   return vDF;
}
```

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Function CreateField(ByVal pColumn As MetaColumn, ByVal pArgs As _
      FieldGenerationArgs(Of DES.DynamicField, DataControlField)) _
   As DES.DynamicField _
   Implements DES.IDataControlFieldGeneratorGuidance.CreateField

   Dim vDF As DES.DynamicField = New DES.DynamicField(pColumn.Name)
   Select Case pArgs.TableName
      Case "Employees"

         Select Case pColumn.Name
            Case "LastName"
               vDF.EditPropertyList = "DataControlWidth='200px'"
         End Select Case

   End Select Case

   Return vDF
End Function
```

MORE SAMPLES ON THE NEXT PAGE

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

Example: Using CreateField to override the scaffolding

The automatic field generation includes columns that do not specify `ScaffoldColumn(false)` on the EntityTable Metadata class. You can use `CreateField()` to omit columns instead.

First in `BeforeGeneration()`, set **pArgs.UseScaffoldColumnAttribute** to `false`. Then use `CreateField()` to create a DynamicField on the desired columns.

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public void BeforeGeneration(
   DES.FieldGenerationArgs<DES.DynamicField, DataControlField> pArgs)
{
   pArgs.UseScaffoldColumnAttribute = false;
}

public DES.DynamicField CreateField(MetaColumn pColumn,
   FieldGenerationArgs<DES.DynamicField, DataControlField> pArgs)
{
   switch (pArgs.TableName)
   {
      case "Employees":
         switch (pArgs.ColumnName)
         {
            case "PhotoPath":
            case "Extension":
            case "Orders":
            case "Employees":
               return null;      // skip these
            default:
               break;
         } // switch
         break;
   default:
      break;
   } // switch

   return new DES.DynamicField(pColumn.Name);
}
```

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

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Sub BeforeGeneration(ByVal pArgs As _
  DES.FieldGenerationArgs(Of DES.DynamicField, DataControlField)) _
    Implements DES.IDataControlFieldGeneratorGuidance.BeforeGeneration

  pArgs.UseScaffoldColumnAttribute = False
End Sub

Public Function CreateField(ByVal pColumn As MetaColumn, ByVal pArgs As _
    FieldGenerationArgs(Of DES.DynamicField, DataControlField)) _
  As DES.DynamicField _
  Implements DES.IDataControlFieldGeneratorGuidance.CreateField

  Select Case pArgs.TableName
    Case "Employees"
      Select Case pColumn.Name
        Case "PhotoPath", "Extension", "Orders", "Employees"
          Return Nothing    ' skip these
      End Select Case
  End Select Case

  Return New DES.DynamicField(pColumn.Name)
End Function
```

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

IDataControlFieldGeneratorGuidance.AfterGeneration method

Called after automatic field generation to let you modify the **pArgs.Fields** collection. *It is not called if automatic field generation is disabled.* You will take advantage of several methods on pArgs to assist you.

- Remove a column by calling `pArgs.HideFieldByColumnName()`.
- Add a column by calling `pArgs.AddColumn()` or `AddField()`.
- Edit a column's properties by retrieving it with `pArgs.GetFieldByColumnName()`.
- Change the order of a column by calling `pArgs.MoveFieldByColumnName()`. There are two versions. One takes the column name after which the column is moved. The other takes a numeric position for the column.

If you don't need to take these actions, still define `AfterGeneration()`, leaving it an empty method. Otherwise modify the pArgs parameter.

Example: Modify pArgs.Fields

For the Employees table, when showing the Details usage, hide these columns: PhotoPath, Employees, and Orders.

For the Employees table, when in Edit or Insert mode, make the PhotoPath's textbox wider.

For the Customers table, rearrange the columns.

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public void AfterGeneration(
   DES.FieldGenerationArgs<DES.DynamicField, DataControlField> pArgs)
{
   switch (pArgs.TableName)
   {
      case "Employees":
         if (pArgs.DataControlAction == DES.DataControlAction.Details)
            {
               pArgs.HideFieldByColumnName("PhotoPath");
               pArgs.HideFieldByColumnName("Employees");
               pArgs.HideFieldByColumnName("Orders");
            }
            else if (pArgs.DataBoundControlMode != DataBoundControlMode.ReadOnly)
            {
               DES.DynamicField vDF =
                  pArgs.GetFieldByColumnName("PhotoPath") as DES.DynamicField;
               vDF.EditPropertyList = "DataControlWidth='300px'";
               vDF.InsertPropertyList = "DataControlWidth='300px'";
            }
            break;

      case "Customers":
         pArgs.MoveFieldByColumnName("CompanyName", "ContactName");
         pArgs.MoveFieldByColumnName("ContactTitle", "ContactName");
         pArgs.MoveFieldByColumnName("Country", 999, true); // last
         pArgs.MoveFieldByColumnName("Phone", 1, false); // move relative by 1
         pArgs.MoveFieldByColumnName("CustomerID", 4, true); // column 5
         break;

      default:
         break;
   } // switch
}
```

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

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Sub AfterGeneration(ByVal pArgs As _
  DES.FieldGenerationArgs(Of DES.DynamicField, DataControlField)) _
    Implements DES.IDataControlFieldGeneratorGuidance.BeforeGeneration

  Select Case pArgs.TableName
    Case "Employees"
      If pArgs.DataControlAction = DES.DataControlAction.Details Then
        pArgs.HideFieldByColumnName("PhotoPath")
        pArgs.HideFieldByColumnName("Employees")
        pArgs.HideFieldByColumnName("Orders")
      Else If pArgs.DataBoundControlMode <> DataBoundControlMode.ReadOnly Then
        Dim vDF As DES.DynamicField = _
          CType(pArgs.GetFieldByColumnName("PhotoPath"), DES.DynamicField)
        vDF.EditPropertyList = "DataControlWidth='300px'"
        vDF.InsertPropertyList = "DataControlWidth='300px'"
      End If

    case "Customers"
      pArgs.MoveFieldByColumnName("CompanyName", "ContactName")
      pArgs.MoveFieldByColumnName("ContactTitle", "ContactName")
      pArgs.MoveFieldByColumnName("Country", 999, True) ' last
      pArgs.MoveFieldByColumnName("Phone", 1, False) ' move relative by 1
      pArgs.MoveFieldByColumnName("CustomerID", 4, True) ' column 5
  End Select Case
End Sub
```

Example: Do nothing with AfterGeneration

Just declare the empty method.

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public void AfterGeneration(
  DES.FieldGenerationArgs<DES.DynamicField, DataControlField> pArgs)
{
}
```

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Sub AfterGeneration(ByVal pArgs As _
  DES.FieldGenerationArgs(Of DES.DynamicField, DataControlField)) _
    Implements DES.IDataControlFieldGeneratorGuidance.BeforeGeneration

End Sub
```

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

PeterBlum.DES.DynamicData.FieldGenerationArgs

The IPatternTemplateFieldGeneratorGuidance and IDataControlTemplateFieldGeneratorGuidance interfaces take the **FieldGenerationArgs class** as a parameter for each of its methods. Use it to establish the **Fields** collection, a list of field objectsthat will be shown in the DataBound controls.

FieldGenerationArgs Properties

- **Fields** (List<System.Web.UI.WebControls.DataControlField>) – Add and modify elements of this collection to establish the DataControlFields used on the GridView or DetailsView control.

  It accepts all types of DataControlFields. You will use the DynamicField for columns supported by DynamicData. Use other types, including TemplateField, to support other situations.

  While you can edit this collection directly, use these methods to make things easier:

  - AddColumn() method – Create a `PeterBlum.DES.DynamicData.DynamicField` for a column name.
  - AddField() method – Add any instance of a DataControlField subclass, such as ButtonField, TemplateField, and DynamicField.
  - GetFieldByColumnName() method – Retrieves a DynamicField by the column name. Use it to edit the properties of the DynamicField. Usually called in the `AfterGeneration()` method.
  - HideFieldByColumnName() method – Removes a DataControlField that was already added. Usually called in the `AfterGeneration()` method.
  - MoveFieldByColumnName() method – Changes the position of a column name in the list. Usually called in the `AfterGeneration()` method.

- **HasFields** (bool) – When `true`, the Fields collection has DataControlField elements. When `false`, it does not.
- **Table** (System.Web.DynamicData.MetaTable) – *ReadOnly.* Identifies the table that is being displayed.
- **TableName** (string) – *ReadOnly.* The table name.
- **DataBoundControlAdapter** (PeterBlum.DES.DynamicData.BaseDataBoundControlAdapter) – *ReadOnly.* Identifies the DataBound control in its **DataBoundControl** property. See "The BaseDataBoundControlAdapter class".
- **DataBoundControlType** (enum PeterBlum.DES.DynamicData.DataBoundControlType) – *ReadOnly.* Determines if the DataBound control is list or single record style.

  Useful in defining different field lists based on the type of control.

  The enumerated type `PeterBlum.DES.DynamicData.DataBoundControlType` has these values:

  - `List`
  - `Single`

- **DataBoundControlMode** (enum System.Web.UI.WebControls.DataBoundControlMode) – *ReadOnly.* The current mode of the DataBound control.

  For lists, it is either `ReadOnly` or `Edit`. When `Edit`, a single row is in Edit mode. All others rows are `ReadOnly`.

  For single record controls, it can be any mode (`ReadOnly`, `Edit`, or `Insert`).

- **DataControlAction** (enum PeterBlum.DES.DynamicData.DataControlAction) – *ReadOnly.* Identifies the usage ("action") for which the DataBound control is used.

  The enumerated type `PeterBlum.DES.DynamicData.DataControlAction` has these values:

  - `List` – List style control including grids. For example **List.aspx** Page Template.
  - `Details` – Single record that is not editable. For example **Details.aspx** Page Template.
  - `Edit` – Single record that is in Edit mode. For example **Edit.aspx** Page Template.
  - `Insert` – Single record that is in Insert mode. For example **Insert.aspx** Page Template.

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

- **UseAutoGenerate** (enum PeterBlum.DES.DynamicData.UseAutoGenerate) – Use within the `BeforeGeneration()` method to indicate if automatic field generation will occur. When it does not occur, your `CreateField()` and `AfterGeneration()` methods will not be called. You are expected to populate the **Fields** collection within BeforeGeneration().

  The enumerated type `PeterBlum.DES.DynamicData.UseAutoGenerate` has these values:

  - `No` - Do not use automatic fields generation. All generation already occurred in your `BeforeGeneration()` method.
  - `Yes` - Use automatic fields generation to populate the **Fields** collection. Use your `CreateField()` and `AfterGeneration()` methods to modify how it works.
  - `HasNoFields` - When automatic fields generation when no DataControlFields have been added to the Fields property during `BeforeGeneration()`. This is the default.

- **UseScaffoldColumnAttribute** (boolean) – Determines if columns are added using the ScaffoldColumnAttribute.

  When `true`, columns are added using the ScaffoldColumnAttribute. They are also determined by the `CreateField()` method.

  When `false`, the following still determine if a control is created:

  - `CreateField()` method
  - **ShowLongStringsInLists**, **ShowChildrenTemplateInEditMode**, **ShowReadOnlyColumnsInEditMode**, and **Categories** properties

  It defaults to `true`.

- **UseScaffoldOrdering** (boolean) – The ScaffoldColumnAttribute controls can specify the ordering of fields through the **Position** or **AfterThisColumnName** properties.

  When `true`, any DES.ScaffoldColumnAttribute that specifies ordering through the **Position** or **AfterThisColumnName** properties will have that ordering applied.

  When `false`, they are ignored.

  It defaults to `true`.

- **ShowLongStringsInLists** (boolean) – A long string is any string column that does not have a maximum length. Since these columns can have very large data, they may create exceptionally large lists of records in the browser. So its a good idea to hide or abbreviate the data.

  When `false`, any column marked IsLongString is not added by the GenerateFields method to list style controls. When `true`, they are. Generally this is set to `true` when your readonly Field Template to show text can truncate long strings.

  It defaults to `false`.

- **ShowReadOnlyColumnsInEditMode** (boolean) – In edit or insert mode, it determines if a column that is strictly readonly will appear.

  A column that is strictly readonly is defined by the associated property on the EntityTable class having only a GET clause.

C#	VB

`public decimal TotalPrice`
`{`
`  get { return Price * Quantity; }`
`}`
	`Public Property TotalPrice As Decimal`
	`  Get`
	`  Return Price * Quantity`
	`  End Get`
	`End Property`

  Set to `true` to add when scaffolding. To `false` to omit them.

  It defaults to `false`.

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

- **ShowChildrenTemplateInEditMode** (boolean) – In edit mode, this determines if the DetailsView will show columns of type MetaChildrenColumn (which uses the **Children.ascx** template). This Field Template provides a hyperlink to show another page with a list of records. It may not make sense to offer this within edit mode.

  Set to `true` to display this Field Template. To `false` to hide it.

  In native ASP.NET Dynamic Data, it is displayed. So this defaults to `true`.

- **Categories** (string) – A list of category names that must be assigned to the columns to include. Use this to narrow your search. Prior to eliminating a data field by category name, it has already been reviewed for elimination by the ScaffoldColumnAttribute and the other rules of the FieldGenerationArgs class.

  Data fields can have CategoryAttributes in their metadata to assist with setting up Automatic Scaffolding and NamedStyles. CategoryAttributes permit a multiple category names. Just one of those names must match just one name in the Categories property in order for the column to be included or excluded.

  Use a pipe delimited list of the category names when multiple names are needed. Use **ExcludeCategories** to determine if the category is included or excluded.

  Matching is case insensitive. When unassigned, the scaffolding does not use categories in matching.

  It defaults to "".

- **ExcludeCategories** (Boolean) – Determines what happens when the **Categories** property contains a matching name to the CategoryAttribute's value. Normally when the Category name is found in the CategoryAttribute, that column is a match. This property can make that column not match.

  When `true`, exclude columns with any of these category names.

  When `false`, include columns with any of these category names.

  It defaults to `false`.

- **CategoriesWithNoMatchAreExcluded** (Boolean) – Determines what happens when the **Categories** property does *not* contain a matching name to the CategoryAttribute's value, including when the data field lacks a CategoryAttribute.

  When `true`, exclude data fields with any of these category names.

  When `false`, include data fields with any of these category names.

  It defaults to `false`.

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

AddColumn() method

Adds a field object for the column name to the end of the **Fields** collection. Typically this is called from the `BeforeGeneration()` method.

When using IPatternTemplateFieldGeneratorGuidance, it adds a `PeterBlum.DES.DynamicData.DataFieldInPattern` object.

When using IDataControlFieldGeneratorGuidance, it adds a `PeterBlum.DES.DynamicData.DynamicField` object.

[C#]

```
PeterBlum.DES.DynamicData.DynamicField AddColumn(string pColumnName)

PeterBlum.DES.DynamicData.DynamicField AddColumn(string pColumnName,
      string pUIHint)

PeterBlum.DES.DynamicData.DynamicField AddColumn(string pColumnName,
      string pUIHint, string pHeaderText, string pSortExpression)
```

[VB]

```
Sub AddColumn(ByVal pColumnName As String)
      As PeterBlum.DES.DynamicData.DynamicField

Sub AddColumn(ByVal pColumnName As String, ByVal pUIHint As String)
      As PeterBlum.DES.DynamicData.DynamicField

Sub AddColumn(ByVal pColumnName As String, ByVal pUIHint As String,
      ByVal pHeaderText As String, ByVal pSortExpression As String)
      As PeterBlum.DES.DynamicData.DynamicField
```

Parameters

pColumnName

The name of the column to add. Use the exact name from the database, not what you assign to any DisplayNameAttribute. The name is case sensitive.

pUIHint

Optional. Default Field Template to use. Assigned to the **DynamicField.UIHint** property.

pHeaderText

Optional. The text shown for a label with this column. Assigned to the **DynamicField.HeaderText** property.

pSortExpression

Optional. Expression used to sort this column. Assigned to the **DynamicField.SortExpression** property.

Returns

The `PeterBlum.DES.DynamicData.DynamicField` instance that was created. Use it to edit more properties of this class.

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

Example

[C#]

```
using DES = PeterBlum.DES.DynamicData;

…

public void BeforeGeneration(
   DES.FieldGenerationArgs<DES.DynamicField, DataControlField> pArgs)
{
   switch (pArgs.TableName)
   {
      case "Employees":
         if (pArgs.DataControlAction == DES.DataControlAction.List)
         {
            pArgs.AddColumn("FirstName");
            pArgs.AddColumn("LastName");
            pArgs.AddColumn("BirthDate", "Calendar");
         }
         break;
    } // switch
}
```

[VB]

```
Import DES = PeterBlum.DES.DynamicData

…

Public Sub BeforeGeneration(ByVal pArgs As _
   DES.FieldGenerationArgs(Of DES.DynamicField, DataControlField)) _
      Implements DES.IDataControlFieldGeneratorGuidance.BeforeGeneration

   Select Case pArgs.TableName
      Case "Employees"
         If pArgs.DataControlAction = DES.DataControlAction.List Then
            pArgs.AddColumn("FirstName")
            pArgs.AddColumn("LastName")
            pArgs.AddColumn("BirthDate", "Calendar")
         End If
    End Select Case
End Sub
```

Example: Setting the column width

By default, the GridView columns are sized automatically. If you want to control the width, set it on the **DynamicField.ItemStyle.Width** property.

You can avoid some word-wrap effects in a column by setting **DynamicField.ItemStyle.Wrap** to `false`.

[C#]

```
PeterBlum.DES.DynamicData.DynamicField vDF = pArgs.AddColumn("FirstName");
vDF.ItemStyle.Width = new Unit("100px");

PeterBlum.DES.DynamicData.DynamicField vDF2 = pArgs.AddColumn("ContactTitle");
vDF2.ItemStyle.Wrap = false;
```

[VB]

```
Dim vDF As PeterBlum.DES.DynamicData.DynamicField = pArgs.AddColumn("FirstName")
vDF.ItemStyle.Width = New Unit("100px")

Dim vDF2 As PeterBlum.DES.DynamicData.DynamicField = pArgs.AddColumn("ContactTitle")
vDF2.ItemStyle.Wrap = False
```

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

AddField() method

Add any field object to the end of the **Fields** collection.

Typically this is called from the `BeforeGeneration()` method.

[C#]

```
DataControlField AddField(DataControlField pField)
DES.BaseItemInPattern AddField(DES.BaseItemInPattern pField)
```

[VB]

```
Function AddField(ByVal pField As DataControlField) As DataControlField
Function AddField(ByVal pField As DES.BaseItemInPattern) As DES.BaseItemInPattern
```

Parameters

pField

The field object. When using IPatternTemplateFieldGeneratorGuidance, pass any subclass of PeterBlum.DES.DynamicData.BaseItemInPattern. Typically you pass a DataFieldInPattern object. When using IDataControlFieldGeneratorGuidance, pass any subclass of DataControlField. Typically you pass a DynamicField.

Returns

The same field instance that was passed in. Allows you to instantiate as the parameter and use the instance after calling this method.

Example: IPatternTemplateFieldGeneratorGuidance

[C#]

```
public void BeforeGeneration(
   DES.FieldGenerationArgs<DES.DataFieldInPattern, DES.BaseItemInPattern> pArgs)
{
   switch (pArgs.TableName)
   {
      case "Employees":
         if (pArgs.DataControlAction == PeterBlum.DES.DynamicData.DataControlAction.List)
         {
            pArgs.AddColumn("FirstName");
            pArgs.AddColumn("LastName");
            pArgs.AddColumn("BirthDate");
            DataFieldListInPattern vButtonField = pArgs.AddField(
               new DataFieldListInPattern("ButtonLink"));
         }
         break;
   default:
      break;
    } // switch
}
```

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

[VB]

```
Public Sub BeforeGeneration(ByVal pArgs As _
   DES.FieldGenerationArgs(Of DES.DataFieldInPattern, DES.BaseItemInPattern))
   Select Case pArgs.TableName
      Case "Employees"
         If pArgs.DataControlAction = DES.DataControlAction.List Then
            pArgs.AddColumn("FirstName")
            pArgs.AddColumn("LastName")
            pArgs.AddColumn("BirthDate")
            Dim vButtonField As DES.DataFieldListInPattern = pArgs.AddField( _
               New DES.DataFieldListInPattern("ButtonLink"))
         End If
    End Select Case
End Sub
```

Example: IDataControlFieldGeneratorGuidance

[C#]

```
public void BeforeGeneration(
   DES.FieldGenerationArgs<DES.DynamicField, DataControlField> pArgs)
{
   switch (pArgs.TableName)
   {
      case "Employees":
         if (pArgs.DataControlAction == PeterBlum.DES.DynamicData.DataControlAction.List)
         {
            pArgs.AddColumn("FirstName");
            pArgs.AddColumn("LastName");
            pArgs.AddColumn("BirthDate");
            ButtonField vButtonField = pArgs.AddField(new ButtonField());
            vButtonField.Text = "Edit";
            vButtonField.CommandName = "Edit";
         }
         break;
   default:
      break;
    } // switch
}
```

[VB]

```
Public Sub BeforeGeneration(ByVal pArgs As _
   DES.FieldGenerationArgs(Of DES.DynamicField, DataControlField))
   Select Case pArgs.TableName
      Case "Employees"
         If pArgs.DataControlAction = DES.DataControlAction.List Then
            pArgs.AddColumn("FirstName")
            pArgs.AddColumn("LastName")
            pArgs.AddColumn("BirthDate")
            Dim vButtonField As ButtonField = pArgs.AddField(New ButtonField())
            vButtonField.Text = "Edit"
            vButtonField.CommandName = "Edit"
         End If
    End Select Case
End Sub
```

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

Example: Setting the column width

By default, the GridView columns are sized automatically. If you want to control the width, set it on the **DataControlField.ItemStyle.Width** property.

You can avoid some word-wrap effects in a column by setting **DataControlField.ItemStyle.Wrap** to `false`.

[C#]

```
DataControlField vDCF = pArgs.AddField(new ButtonField());
vDCF.ItemStyle.Width = new Unit("100px");

DataControlField vDCF2 = pArgs.AddField(new DynamicField("ContactTitle"));
vDCF2.ItemStyle.Wrap = false;
```

[VB]

```
Dim vDCF As DataControlField = pArgs.AddField(New ButtonField())
vDCF.ItemStyle.Width = New Unit("100px")

Dim vDCF2 As DataControlField = pArgs.AddField(New DynamicField("ContactTitle"))
vDCF2.ItemStyle.Wrap = False
```

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

GetFieldByColumnName() method

Get the field object from **Fields** collection that is assigned to the column name you pass. Typically this is called from the `AfterGeneration()` method to access and modify the properties of the DynamicField or DataFieldInPattern objects.

[C#]

```
DataControlField GetFieldByColumnName(string pColumnName)
DataControlField GetFieldByColumnName(string pColumnName, out int pPosition)
DES.BaseItemInPattern GetFieldByColumnName(string pColumnName)
DES.BaseItemInPattern GetFieldByColumnName(string pColumnName, out int pPosition)
```

[VB]

```
Function GetFieldByColumnName(ByVal pColumnName As String) As DataControlField
Function GetFieldByColumnName(ByVal pColumnName As String, _
      ByRef pPosition As Integer) As DataControlField
Function GetFieldByColumnName(ByVal pColumnName As String) As DES.BaseItemInPattern
Function GetFieldByColumnName(ByVal pColumnName As String, _
      ByRef pPosition As Integer) As DES.BaseItemInPattern
```

Parameters

pColumnName

The column name to find. Case sensitive.

pPosition

The position of the field object in the Fields collection. 0 is the first item. If -1, it did not find a match.

Returns

The `PeterBlum.DES.DynamicData.DynamicField` instance or `PeterBlum.DES.DynamicData.DataFieldInPattern` that was found or `null`/`Nothing` if not found.

Example: Using IDataControlFieldGeneratorGuidance

[C#]

```
public void AfterGeneration(
   DES.FieldGenerationArgs<DES.DynamicField, DataControlField> pArgs)
{
   switch (pArgs.TableName)
   {
      case "Employees":
         if (pArgs.DataBoundControlMode != DataBoundControlMode.ReadOnly)
         {
         // widen the PhotoPath textbox
            PeterBlum.DES.DynamicData.DynamicField vDF =
               pArgs.GetFieldByColumnName("PhotoPath")
                  as PeterBlum.DES.DynamicData.DynamicField;
            vDF.EditPropertyList = "DataControlWidth='300px'";
            vDF.InsertPropertyList = "DataControlWidth='300px'";
         }
         break;
   default:
      break;
    } // switch
}
```

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

[VB]

```
Public Sub AfterGeneration(ByVal pArgs As _
   DES.FieldGenerationArgs(Of DES.DynamicField, DataControlField))
   Select Case pArgs.TableName
      Case "Employees"
         If pArgs.DataBoundControlMode <> DataBoundControlMode.ReadOnly Then

         ' widen the PhotoPath textbox
            Dim vDF As PeterBlum.DES.DynamicData.DynamicField = _
               CType(pArgs.GetFieldByColumnName("PhotoPath"), _
                  PeterBlum.DES.DynamicData.DynamicField)
            vDF.EditPropertyList = "DataControlWidth='300px'"
            vDF.InsertPropertyList = "DataControlWidth='300px'"
         End If
    End Select Case
End Sub
```

Example: Setting the column width

By default, the GridView columns are sized automatically. If you want to control the width, set it on the **DataControlField.ItemStyle.Width** property.

You can avoid some word-wrap effects in a column by setting **DataControlField.ItemStyle.Wrap** to `false`.

[C#]

```
DataControlField vDCF = pArgs.GetFieldByColumnName("LastName");
vDCF.ItemStyle.Width = new Unit("100px");

DataControlField vDCF2 = pArgs.GetFieldByColumnName("ContactTitle");
vDCF2.ItemStyle.Wrap = false;
```

[VB]

```
Dim vDCF As DataControlField = pArgs.GetFieldByColumnName("LastName")
vDCF.ItemStyle.Width = New Unit("100px")

Dim vDCF2 As DataControlField = pArgs.GetFieldByColumnName("ContactTitle")
vDCF2.ItemStyle.Wrap = False
```

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

HideFieldByColumnName() method

Prevent the field object assigned to a specific column name in **Fields** collection from appearing. It doesn't delete the element. Instead, it sets its **Visible** or **Enabled** property to `false`. Typically this is called from the `AfterGeneration()` method.

[C#]

```
bool HideFieldByColumnName(string pColumnName)
```

[VB]

```
Function HideFieldByColumnName(ByVal pColumnName As String) As Boolean
```

Parameters

pColumnName

The column name to hide. Case sensitive.

Returns

`True` when found or `false` when not found.

Example: Using IDataControlFieldGeneratorGuidance

[C#]

```
public void AfterGeneration(
   DES.FieldGenerationArgs<DES.DynamicField, DataControlField> pArgs)
{
   switch (pArgs.TableName)
   {
      case "Employees":
         if (pArgs.DataControlAction == DES.DataControlAction.Details)
         {
            pArgs.HideFieldByColumnName("PhotoPath");
         }
         break;
   default:
      break;
    } // switch
}
```

[VB]

```
Public Sub AfterGeneration(ByVal pArgs As _
   DES.FieldGenerationArgs(Of DES.DynamicField, DataControlField))
   Select Case pArgs.TableName
      Case "Employees"
         If pArgs.DataControlAction <> DES.DataControlAction.Details Then

            pArgs.HideFieldByColumnName("PhotoPath")
         End If
    End Select Case
End Sub
```

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

MoveFieldByColumnName() method

Change the position of a field object in the **Fields** collection. There are two methods available. The first shown moves *pColumnNameToMove* after the column specified in the *pAfterThisColumnName* parameter. The second moves it to a numeric position. That position is either a column number or relative to the *pColumnNameToMove*, depending on the *pAbsolutePosition* parameter.

Typically this is called from the `AfterGeneration()`.

[C#]

```
DataControlField MoveFieldByColumnName(string pColumnNameToMove,
      string pAfterThisColumnName)

DataControlField MoveFieldByColumnName(string pColumnNameToMove, int pPosition,
      bool pAbsolutePosition)

DES.BaseItemInPattern MoveFieldByColumnName(string pColumnNameToMove,
      string pAfterThisColumnName)

DES.BaseItemInPattern MoveFieldByColumnName(string pColumnNameToMove,
      int pPosition, bool pAbsolutePosition)
```

[VB]

```
Function MoveFieldByColumnName(ByVal pColumnNameToMove As String,
      ByVal pAfterThisColumnName As String) As DataControlField

Function GetFieldByColumnName(ByVal pColumnNameToMove As String,
      ByVal pPosition As Integer, ByVal pAbsolutePosition As Boolean)
      As DataControlField

Function MoveFieldByColumnName(ByVal pColumnNameToMove As String,
      ByVal pAfterThisColumnName As String) As DES.BaseItemInPattern

Function GetFieldByColumnName(ByVal pColumnNameToMove As String,
      ByVal pPosition As Integer, ByVal pAbsolutePosition As Boolean)
      As DES.BaseItemInPattern
```

Parameters

pColumnNameToMove

The column name to move. Case sensitive. If not found, it throws an exception.

pAfterThisColumnName

Position immediately after this column in the **Fields** collection. Case sensitive. If not found, it throws an exception.

pPosition

The position to move the column in the **Fields** collection. Its use depends on the *pAbsolutePosition* property. When `true`, this is the actual column number where 0 is the first item. Use 999 to position as the last item. When `false`, this is a relative position from the position of *pColumnNameToMove*. For example, -1 moves back one column and 1 moves forward one column.

pAbsolutePosition

Determines how to use the *pPosition* parameter. When `true`, it is an actual column number. When `false`, it is relative to the position of *pColumnNameToMove*.

Returns

The `PeterBlum.DES.DynamicData.DynamicField` or `PeterBlum.DES.DynamicData.DataFieldInPattern` instance that matched to *pColumnNameToMove* so you can edit it further.

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

Example

[C#]

```
public void AfterGeneration(
   DES.FieldGenerationArgs<DES.DynamicField, DataControlField> pArgs)
{
   switch (pArgs.TableName)
   {
      case "Customers":
         pArgs.MoveFieldByColumnName("CompanyName", "ContactName");
         pArgs.MoveFieldByColumnName("Country", 999, true);
         pArgs.MoveFieldByColumnName("Phone", 1, false); // move relative by 1
         pArgs.MoveFieldByColumnName("ContactTitle", "ContactName");
         pArgs.MoveFieldByColumnName("CustomerID", 4, true);
         break;
   default:
      break;
    } // switch
}
```

[VB]

```
Public Sub AfterGeneration(ByVal pArgs As _
   DES.FieldGenerationArgs(Of DES.DynamicField, DataControlField))
   Select Case pArgs.TableName
      Case "Customers"
         pArgs.MoveFieldByColumnName("CompanyName", "ContactName")
         pArgs.MoveFieldByColumnName("Country", 999, true)
         pArgs.MoveFieldByColumnName("Phone", 1, false) ' move relative by 1
         pArgs.MoveFieldByColumnName("ContactTitle", "ContactName")
         pArgs.MoveFieldByColumnName("CustomerID", 4, true)
    End Select Case
End Sub
```

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

Creating your own FieldGenerator Class

ASP.NET Dynamic Data lets you create a FieldGenerator class to implement scaffolding. By default, DES Dynamic Data provides a FieldGenerator class so you don't have to. Its FieldGenerator lets you implement the IPatternTemplateFieldGeneratorGuidance or IDataControlFieldGeneratorGuidance interface on your Page Template so you don't have to create a separate class.

If you prefer, you can create a FieldGenerator class. Here's how.

1. Create a class that implements `System.Web.UI.IAutoFieldGenerator`.
2. Create the `GenerateFields()` method as demanded by the IAutoFieldGenerator interface.
3. Programmatically create your class within the `Page_Init()` method and assign it to the appropriate DataBound Control Adapter on your page's DynamicDataManager control. The base class for all Adapters, `PeterBlum.DES.DynamicData.BaseDataBoundControlAdapter`, has the property **FieldsGenerator**. Assign your object to **FieldsGenerator**.

Example

Please see this article for a real world FieldGenerator class called FilteredFieldsManager: http://csharpbits.notaclue.net/2008/05/dynamicdata-generate-columnsrows-using.html

This is the DynamicDataManager control hosting a DynamicListView control that will use this FieldGenerator.

```
<desDD:DynamicDataManager ID="DynamicDataManager1" runat="server"
   AutoLoadForeignKeys="true" >
   <Adapters>
      <desDD:DynamicListViewAdapter DataBoundControlID="DynamicListView1"
        ValidationSummaryControlID="ValidationSummary1"
        SupportsEditMode="true" SupportsInsertMode="true" />
   </Adapters>
</desDD:DynamicDataManager>
```

Here is the code within the DynamicDataManager's Init event handler or `Page_Init()` method that updates the DynamicListViewAdapter shown above with the FilteredFieldsManager object.

[C#]

```
FilteredFieldsManager vFieldsGenerator =
  new FilteredFieldsManager(MyDataSource.GetTable());
DynamicDataManager1.Adapters[0].FieldsGenerator = vFieldsGenerator;
```

[VB]

```
Dim vFieldsGenerator As FilteredFieldsManager = _
  New FilteredFieldsManager(MyDataSource.GetTable())
DynamicDataManager1.Adapters(0).FieldsGenerator = vFieldsGenerator
```

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

Customizing the Master Page

The default Page Templates all use a Master Page to provide a consistent interface. You will probably modify or replace the Master Page and its style sheet file so it conforms with your web application. ASP.NET Dynamic Data adds the file **Site.master** and **Site.css** to your **[web application]** folder. These are the default Master Page and its style sheet file.

Click on any of these topics to jump to them:

- Elements of a Master Page
- Renaming the Master Page
- Change the appearance of the Master Page
- Change the style sheet file
- Selecting between several Master Pages on a PageTemplate

Elements of a Master Page

The default **Site.master** file has these elements.

- Reference to a style sheet file using a `<link>` tag in the `<head>` section.
- ScriptManager control.
- ContentPlaceHolder with ID="ContentPlaceHolder1".

DES Dynamic Data has you modify that file to add:

- PageManager control with its **AJAXFramework** property set to "`MicrosoftAJAX`".

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

  For details on the PageManager, see the **General Features Guide**.

  If you want all validators to use a different ErrorFormatter, use the **DefaultErrorFormatterSkinID** property. This example uses the PopupViewErrorFormatter with its **Display** property set to `Static`. Learn more about this property in the **Validation User's Guide**.

  ```
  <des:PageManager ID="PageManager1" runat="server" AJAXFramework="MicrosoftAJAX"
    DefaultErrorFormatterSkinID="PopupView {Display='Static'}" />
  ```

- Customizer control, which is optional but recommended as its properties are so frequently edited.

  ```
  <desDD:Customizer ID="Customizer1" runat="server" />
  ```

Note: DES Dynamic Data provides 3 additional style sheet files in the ~/DynamicData/Content/ folder. The <link> tags to these files are automatically added into the same <link> tag that DES uses for its other style sheet files.

Any other element of the default **Site.master** file can be modified or removed.

Sample files for a Master Page can be found here:

[C#]

[DES Product Folder]/DynamicData/CS/Samples/SiteMaster.aspx

[VB]

[DES Product Folder]/DynamicData/VB/Samples/SiteMaster.aspx

EXAMPLE CODE SHOWN ON THE NEXT PAGE

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

Example: Minimal Master Page

```
<%@ Master Language="C#" CodeFile="Site.master.cs" Inherits="Site" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
  <head runat="server">
    <link href="~/Site.css" rel="stylesheet" type="text/css" />
  </head>

<body class="template" >
  <form id="form1" runat="server">
    <asp:ScriptManager ID="ScriptManager1" runat="server" />

    <des:PageManager ID="DESPageManager" runat="server"
      AJAXFramework="MicrosoftAJAX" />
    <desDD:Customizer ID="Customizer1" runat="server" />

    <asp:ContentPlaceHolder id="ContentPlaceHolder1" runat="server">
    </asp:ContentPlaceHolder>
  </form>
</body>
</html>
```

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

Renaming the Master Page

The file name **Site.master** is often not suitable for a web application that offers pages outside of the ASP.NET Dynamic Data Page Templates. You may want to rename the file. When you do, make these changes:

- Change the file name of the code behind file too. If you also have a "designer" file, rename it too. If you are using Visual Studio, renaming the master page file in Solution Explorer does this for you.

  site.master → ***newname*.master**

  site.master.cs → ***newname*.master.cs** (or .vb)

  site.master.designer.cs → ***newname*.master.designer.cs** (or .vb)

- Update the `<%@ Master %>` tag to reflect the new file name in the **CodeFile** property.

```
<%@ Master Language="C#" CodeFile="newname.master.cs" Inherits="Site" %>
```

- Consider renaming the class name in the code behind file from "Site" to the new file name. If you do, also update the `<%@ Master %>` tag to reflect the new class name in the **Inherits** property.

  [C#]

```
public partial class NewName : System.Web.UI.MasterPage
```

```
<%@ Master Language="C#" CodeFile="newname.master.cs" Inherits="NewName" %>
```

  [VB]

```
Partial Class NewName
    Inherits System.Web.UI.MasterPage
```

```
<%@ Master Language="VB" CodeFile="newname.master.vb" Inherits="NewName" %>
```

- Go through every Page Template, both in the **[web application]/DynamicData/PageTemplates** and **[web application]/DynamicData/CustomPages** folders, changing the **MasterPage** property in `<%@ Page %>` to reflect the new name.

```
<%@ Page Language="C#" MasterPageFile="~/NewName.master"
  CodeFile="Details.aspx.vb" Inherits="Details" %>
```

- Edit the Entry Point Page (**Default.aspx**) changing the **MasterPage** property in `<%@ Page %>` to reflect the new name.

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

Change the appearance of the Master Page

So long as the Elements of a Master Page are preserved, you can fully customize the Master Page.

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

Change the style sheet file

Open the following files to explore how **Site.css** is applied.

- **[web application]/Site.css**
- **[web application]/Site.master**
- **[web application]/DynamicData/PageTemplates/Details.aspx**
- **[web application]/DynamicData/PageTemplates/List.aspx**

Site.css defines just a few "style sheet class names", but has numerous ways to apply them based on the HTML tag on the page.

Class Name	Location and usage
template	Applied to the `<body>` tag in the Master Page. It impacts nearly every element that isn't covered by another class name. Because it includes TABLE, TR, TD and TH elements, every table on the page is affected. That may be excessive as you or web controls you use may also use tables.
allcaps	Master Page uses this on the Dynamic Data Site title.
back	Master Page uses this on the Back hyperlink just under the title
gridview	Applied to the CssClass property of GridViews
even	Used by GridView on alternating rows
PageXofN	Used by the **[web app]/DynamicData/Content/GridViewPager.ascx** user control
Results	Used by the **[web app]/DynamicData/Content/GridViewPager.ascx** user control

Handling Conflicts between style sheets and web controls

ALERT: This section applies to the Site.css style sheet file supplied with the Dynamic Data 1.0 RTM. Later versions have modified Site.css and may have less or no conflicts.

Unfortunately, the `template` and `gridview` classes affect HTML elements so broadly, that they can impact web controls and HTML not intended to be affected in GridView and DetailsView controls. This includes some DES web controls that use tables. For example, the Calendar and PopupViews have excessive space around many cells because of padding styles. Plus the background color and fonts are incorrect.

Note: Conflicts are found in the GridView and DetailsView controls. If you are using FormView, ListView, DynamicFormView, or DynamicListView, you can avoid the conflicts by assigning style sheets to the <td> tags that are contained. The GridView and DetailsView use inheritance to apply specific styles to their <td> tags and that inheritance is the source of conflicts.

You will have to customize the style sheets further to avoid conflicts. DES has done this for its controls with the **[web application]/DynamicData/Content/FixSiteforDD1.css** style sheet file. Add this line to `Page_Load()` of pages with conflicts:

```
PeterBlum.DES.Globals.Page.StyleSheetManager.Include(
     PeterBlum.DES.StyleSheetFiles.FixSiteforDD1)
```

Note: This line is already included in the default Page Template forms.

SEE THE NEXT PAGE FOR GUIDELINES TO FIX YOUR OWN STYLE CONFLICTS

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

Here are guidelines to fixing your own style conflicts. *Suggestion: Use the ~/DynamicData/Content/FixSiteforDD1.css file as a source for examples.*

1. These classes are the cause of most conflicts. Your task is to override them on a case-by-case basis.

```
body.template table.listview th, table.gridview th, table.detailstable th,
body.template table.listview td, table.gridview td, table.detailstable td
{
    padding-top: 6px;
    padding-bottom: 6px;
    padding-left: 10px;
    padding-right: 10px;
    line-height: 1.1em;
}
body.template table.listview td, table.gridview td, table.detailstable td
{
    border-left: 0px;
    border-right: 0px;
    border-bottom: 1px dotted #dadada;
}
```

2. Locate an element that appears incorrect, such as has too much padding, an unexpected border or the wrong background.
3. Make sure it has its own style sheet class defined on the elements that are problematic. That class should attempt to remedy the problems imposed by the body.template styles.
4. View the HTML source and locate the actual HTML element that is a problem. Note the tag type, such as , <tr>, or .
5. Duplicate the style sheet class for that tag and change the name of that duplicate using this format:

```
body.template table.listview [html tag name].YourClassName
```

Example

Suppose the `<tr>` element of a List control has the class ListFooter and it has two problems: background color is incorrect and it has too much padding.

```
<tr class="ListFooter" >
```

```
body.template table.listview tr.ListFooter
{
   background-color: white;
   padding: 0px;
}
```

6. Where you cannot directly assign a style sheet class to an HTML element, you may be able to use the style sheet class of a containing element, such as if <table> has the class, you can also affect its contained <tr>, <td>, and <th> tags. Here is the format:

```
body.template table.listview [html tag name].YourClassName [nested html tag name]
```

Example

Suppose the `<tr>` element of a List control has the class ListFooter and it's <td> rows have two problems: background color is incorrect and it has too much padding.

```
<tr class="ListFooter" ><td>
```

```
body.template table.listview tr.ListFooter td
{
   background-color: white;
   padding: 0px;
}
```

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

Selecting between several Master Pages on a PageTemplate

ASP.NET lets you programmatically assign a Master Page file during the Page's PreInit phase. You can take advantage of this to let your Page Templates choose from different Master Pages.

1. Create several master pages.
2. Open a Page Template that will support these Master Pages.
3. Add the `Page_PreInit()` method and write your logic to select a Master Page. Assign the file name to the **Page.MasterPageFile** property.
4. Repeat for each Page Template.

Example

Suppose your site tracks a user's language in the Session collection under the key "Language". You use the cultureinfo naming for languages, such as "en-US" and "fr-FR".

[C#]

```
void Page_PreInit(object sender, EventArgs e)
{
  if (Session["language"] != null)
  {
    switch ((string) Session["language"])
      case "en-US":
        this.MasterPageFile = "~/Site_en_US.master";
        break;
      case "fr-Fr":
        this.MasterPageFile = "~/Site_fr_FR.master";
        break;
    } // switch
  }
}
```

[VB]

```
Private Sub Page_PreInit(ByVal sender As Object, ByVal e As EventArgs)
    If (Not (Session("language")) Is Nothing) Then
        Select Case (CType(Session("language"), String))
            Case "en-US"
                Me.MasterPageFile = "~/Site_en_US.master"
            Case "fr-Fr"
                Me.MasterPageFile = "~/Site_fr_FR.master"
        End Select
    End If
End Sub
```

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

Customizing the Entry Point Page

The **Entry Point Page** provides a list of available tables as declared in the Data Context. When the user clicks on a table name, it goes to the list page for that table. It serves as a home page for a Page Template-based application.

You don't need to use an Entry Point Page at all. You can have any existing page redirect to the list page. See "Going to a specific table".

The default Entry Point Page, **Default.aspx**, uses the same Master Page and style sheet file as Page Templates. You will probably modify or replace the Entry Point Page and its style sheet file so it conforms with your web application.

Click on any of these topics to jump to them:

- Elements of an Entry Point Page
- Renaming the Entry Point Page
- Going to a specific table

Elements of an Entry Point Page

The default **Default.aspx** file has these elements.

- GridView control that provides hyperlinks using table names and URLs determined by the Routing system.
- `Page_Load()` method that establishes the DataSource from the MetaModel class and calls `DataBind()`.

DES Dynamic Data does not require any additional features on this page:

Example: Minimal Entry Point Page

Note: Prerelease versions of ASP.NET 4 replace the HyperLink with a DynamicHyperLink. They also change the style sheets.

```
<%@ Page Language="C#" MasterPageFile="~/Site.master"
   CodeFile="Default.aspx.cs" Inherits="_Default" %>

<asp:Content ID="Content1" ContentPlaceHolderID="ContentPlaceHolder1" Runat="Server">
   <asp:ScriptManagerProxy ID="ScriptManagerProxy1" runat="server" />

   <asp:GridView ID="Menu1" runat="server" AutoGenerateColumns="false"
      CssClass="gridview" AlternatingRowStyle-CssClass="even">
      <Columns>
         <asp:TemplateField HeaderText="Table Name" SortExpression="TableName">
            <ItemTemplate>
               <asp:HyperLink ID="HyperLink1" runat="server"
                  NavigateUrl='<%#Eval("ListActionPath") %>'>
                  <%#Eval("DisplayName") %>
               </asp:HyperLink>
            </ItemTemplate>
         </asp:TemplateField>
      </Columns>
   </asp:GridView>
</asp:Content>
```

[C#]

```
protected void Page_Load(object sender, EventArgs e) {
   System.Collections.IList visibleTables = MetaModel.Default.VisibleTables;
   if (visibleTables.Count == 0)
      throw new InvalidOperationException("There are no accessible tables.");
   Menu1.DataSource = visibleTables;
   Menu1.DataBind();
}
```

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

[VB]

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
  Dim visibleTables As System.Collections.IList = MetaModel.Default.VisibleTables
  If (visibleTables.Count = 0) Then
    Throw New InvalidOperationException("There are no accessible tables.")
  End If
  Menu1.DataSource = visibleTables
  Menu1.DataBind
End Sub
```

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

Renaming the Entry Point Page

The file name **Default.aspx** is often not suitable for a web application that offers pages outside of the ASP.NET Dynamic Data Page Templates. You may want to rename the file. When you do, make these changes:

- Change the file name of the code behind file too. If you also have a "designer" file, rename it too. If you are using Visual Studio, renaming the master page file in Solution Explorer does this for you.

  Default.aspx → ***newname*.aspx**

  Default.aspx.cs → ***newname*.aspx.cs** (or .vb)

  Default.aspx.designer.cs → ***newname*.aspx.designer.cs** (or .vb)

- Update the `<%@ Page %>` tag to reflect the new file name in the **CodeFile** property.

```
<%@ Page Language="C#" CodeFile="newname.aspx.cs" Inherits="Site" %>
```

- Consider renaming the class name in the code behind file from "_Default" to the new file name. If you do, also update the `<%@ Page %>` tag to reflect the new class name in the **Inherits** property.

  [C#]

```
public partial class NewName : System.Web.UI.Page
```

```
<%@ Page Language="C#" CodeFile="newname.aspx.cs" Inherits="NewName" %>
```

  [VB]

```
Partial Class NewName
    Inherits System.Web.UI.Page
```

```
<%@ Page Language="VB" CodeFile="newname.aspx.vb" Inherits="NewName" %>
```

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

Going to a specific table

You can redirect to any table's List page from within your web application by using this code:

[C#]

```
MetaTable vMT = MetaModel.Default.GetTable("TableName");
Response.Redirect(vMT.ListActionPath);
```

[VB]

```
Dim vMT As MetaTable  = MetaModel.Default.GetTable("TableName")
Response.Redirect(vMT.ListActionPath)
```

The `GetTable()` method also takes the class type for an EntityTable class, like `typeof(Products)`/ `GetType(Products)`.

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

Changing the Default paths for PageTemplates

There are cases where you may need Page Template web forms to work with different data sources. Consider setting up two MetaModels. One works with LinqToSQL. The other works with "POCO Entities for Dynamic Data".

When will this happen? You may have existing code that uses the POCO Entity approach while you are working on a new database that uses LinqToSQL.

Page Template files include a data source control specific to the MetaModel for which it was intended. So in this case, you need two versions of the Page Templates folder.

The paths to the Page Templates and CustomPages folder can be changed by using the `PeterBlum.DES.DynamicData.DESDynamicDataRouteHandler` class when you setup the Url Routing for your MetaModel. Its constructor takes two parameters, the path to the Page Templates folder and the path to the CustomPages folder.

```
new DESDynamicDataRouteHandler("PageTemplatePath", "CustomPagePath")
```

Each of these parameters takes tokens which are replaced by components at runtime.

For PageTemplatePath:

- {0} – The folders above the Page Templates folder.
- {1} – The Action. This goes where the name of the web form goes, excluding the ".aspx".

Example: `"{0}PageTemplates_POCOEntities/{1}.aspx"`

For CustomPagePath:

- {0} – The folders above the Page Templates folder.
- {1} – The Table name. It is usually a subfolder of the CustomPages folder.
- {2} – The Action. This goes where the name of the web form goes, excluding the ".aspx".

Example: `"{0}CustomPages_POCOEntities/{1}/{2}.aspx"`

Steps for changing paths

1. Create a new folder within the **[web application]\DynamicData** folder. Since it will contain Page Templates, it probably should use the name "PageTemplates" plus some modifier like "PageTemplates for POCO".
2. Do the same for the CustomPages folder.
3. Create your Page Template files in those folders.
4. In `Application_Start()`, setup the Url Routing for the MetaModel that uses these new folders like this:

   [C#]

```
using DES=PeterBlum.DES.DynamicData;
...
DES.MetaModel vMetaModel = new DES.DESMetaModel();
vMetaModel.RegisterContext(typeof(DataContext),
  new ContextConfiguration() { ScaffoldAllTables = true });
routes.Add(new DynamicDataRoute("{table}/{action}.aspx") {
    Constraints = new RouteValueDictionary(
       new { action = "List|Details|Edit|Insert" }),
    RouteHandler= new DES.DESDynamicDataRouteHandler(
       "{0}PageTemplates_POCOEntities/{1}.aspx",
       "{0}CustomPages_POCOEntities/{1}/{2}.aspx"),
    Model = vMetaModel
});
```

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

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
...
Dim vMetaModel As DES.MetaModel = New DES.DESMetaModel()
vMetaModel.RegisterContext(GetType(DataContext),  _
  New ContextConfiguration() With { .ScaffoldAllTables = True })
routes.Add(New DynamicDataRoute("{table}/{action}.aspx") With { _
  .Constraints = New RouteValueDictionary( _
    New With {.Action = "List|Details|Edit|Insert"}), _
  .RouteHandler= new DES.DESDynamicDataRouteHandler( _
       "{0}PageTemplates_POCOEntities/{1}.aspx", _
       "{0}CustomPages_POCOEntities/{1}/{2}.aspx"),  _
  .Model = vMetaModel
})
```

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

Setting up the DataBound Control at Runtime

When using Page Templates, a good strategy is to avoid putting too much application specific code within them. You want to avoid:

- The maintenance throughout many web form files, including in the **CustomPages** folder.
- Changing a Page Template that is used in several situations, with a different master page.

DES Dynamic Data makes it easy to move that code into the MasterPage. You implement various interfaces on the MasterPage and setup your code there. The interfaces are:

- IUICustomizeColumn – Modify attributes associated with the MetaColumn based on the data of the current row being created. See "Implementing the IUICustomizeColumn Interface".
- ICustomizeFieldTemplate – Modify the controls on a Field Template. See "Implementing the ICustomizeFieldTemplate Interface".
- ICustomizeDataBoundControl – Establish event handlers and set properties on the DataBound control. See "Implementing the ICustomizeDataBoundControl Interface".
- IPatternTemplateFieldGeneratorGuidance or IDataControlFieldGeneratorGuidance – Determine the list of columns that are created by automatic scaffolding. See "Customizing Automatic Scaffolding".

These interfaces work on the Page and UserControls too but the MasterPage is recommended when using Page Templates.

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

Implementing the ICustomizeDataBoundControl Interface

The `PeterBlum.DES.DynamicData.ICustomizeDataBoundControl` Interface lets you interact with the DataBound control on the current Page Template. (It works if you aren't using Page Templates too). It is called during the Page_Init phase of the page cycle, before most events and properties on your DataBound control are used.

Here are some uses of this method:

- Customize a template
- Add a Pager
- Attach to the RowCreated or ItemCreated event with methods that will modify the controls just created.

Click on any of these topics to jump to them:

- Where to implement ICustomizeDataBoundControl
- Adding the ICustomizeDataBoundControl Interface
- ICustomizeDataBoundControl.CustomizeDataBoundControl method
- Util.GetTableName method
- Util.GetFieldTemplate method
- Util.GetTable method

Where to implement ICustomizeDataBoundControl

As an interface, you attach it to a class then create the list of methods within that class. You already have some classes created where you should consider attaching it: the Page, the MasterPage, or any UserControl containing the DynamicDataManager control. If you use any of these, DES will find and use it. You can also explicitly assign the class to the Customizer control's **CustomizeDataBoundControlOwner** property.

How DES locates your interface

You may have this interface declared in a few classes, such as both the MasterPage and UserControl containing a DynamicDataManager. If you let DES search for the interface, it always searches in this order:

1. User control with DynamicDataManager
2. Master Page
3. Page

How to specify which object to use

If you need to specify which of these to use, assign the object to the Customizer control's **CustomizeDataBoundControlOwner** property.

Always set the **CustomizeDataBoundControlOwner** property in the DynamicDataManager's Init event handler method or `Page_Init()` method.

[C#]

```
Customizer1.CustomizeDataBoundControlOwner = this;
```

[VB]

```
Customizer1.CustomizeDataBoundControlOwner = Me
```

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

Adding the ICustomizeDataBoundControl Interface

1. Start by using the `PeterBlum.DES.DynamicData` namespace. It's easiest to map it to the name "DES" just like was done in the code file containing the EntityTable Metadata Classes.

   [C#]

   ```
   using DES = PeterBlum.DES.DynamicData;
   ```

   [VB]

   ```
   Import DES = PeterBlum.DES.DynamicData
   ```

2. Assign the ICustomizeDataBoundControl interface to the desired class.

   [C#]

   ```
   public class WebForm1 : System.Web.UI.WebControls.Page,
       DES.ICustomizeDataBoundControl
   ```

   [VB]

   ```
   Public Class WebForm1
      Implements System.Web.UI.WebControls.Page, _
      Implements DES.ICustomizeDataBoundControl
   ```

3. Implement the `CustomizeDataBoundControl()` method. See "ICustomizeDataBoundControl.CustomizeDataBoundControl method".

   If you hookup to the ItemCreated or RowCreated event and want to use them to modify the Field Templates, first consider using the ICustomizeFieldTemplate interface. See "Implementing the ICustomizeFieldTemplate Interface". Otherwise, DES provides you with these static/Shared methods on the `PeterBlum.DES.DynamicData.Util` class:

   - `GetTableName()` – Gets the table name associated with the DataBound control. If the table name matches, use the overloaded `GetFieldTemplate()` method to get the Field Template for a specific column. See "Util.GetTableName method".
   - `GetFieldTemplate()` – An overloaded method that gets the desired Field Template by specifying the column name and optionally the table name. See "Util.GetFieldTemplate method".

4. If its not already on the page, add the Customizer control. It is the conduit between your method and the code that invokes it. When using Page Templates, typically the Customizer is added to the MasterPage.

5. DES Dynamic Data will discover your control or page class that implements ICustomizeDataBoundControl so long as that class is a User Control that contains the DES DynamicDataManager control, the MasterPage, or the Page. See "How DES locates your interface".

   If it cannot be discovered, assign the object to the Customizer's **CustomizeDataBoundControlOwner** property in either the DynamicDataManager's Init event handler or the `Page_Init()` method like this:

   [C#]

   ```
   Customizer1.CustomizeDataBoundControlOwner = this;
   ```

   [VB]

   ```
   Customizer1.CustomizeDataBoundControlOwner = Me
   ```

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

ICustomizeDataBoundControl.CustomizeDataBoundControl method

The only method on the ICustomizeDataBoundControl Interface. It is called during the Page_Init phase to allow your code to modify the DataBound control in use. Generally you establish event handlers and edit the properties of the DataBound control.

[C#]

```
public void CustomizeDataBoundControl(object pSender,
   EventArgs pArgs)
```

[VB]

```
Public Sub CustomizeDataBoundControl(ByVal pSender As Object,
   ByVal pArgs As EventArgs)
```

Parameters

pSender

A reference to the DataBound control. Typecast to the actual control type before accessing its properties and events.

pArgs

Not used.

Example

Sets up the ItemCreated event handler on a DetailsView. The event's method will edit the Field Template associated with the ContactName column of the Customers table.

[C#]

```
public partial class Site : System.Web.UI.MasterPage,
    DES.ICustomizeDataBoundControl
{
   public void CustomizeDataBoundControl(object pSender, EventArgs pArgs)
   {
      DetailsView vDetailsView = pSender as DetailsView;
      if (pSender != null)
         vDetailsView.ItemCreated += new EventHandler(OnDetailsViewItemCreated);
    }  // CustomizeDataBoundControl

   protected virtual void OnDetailsViewItemCreated(object pSender, EventArgs pArgs)
   {
      DetailsView vDetailsView = pSender as DetailsView;
      DES.TextEditFTUC vTEFieldTemplate =
         DES.Util.GetFieldTemplate(vDetailsView, "Customers", "ContactName")
              as DES.TextEditFTUC;
      if (vTEFieldTemplate != null)
         vTEFieldTemplate.TextBox.BackColor = System.Drawing.Color.Azure;
    } // OnDetailsViewItemCreated
} // class Site
```

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

[VB]

```
Public Class Site
  Inherits System.Web.UI.MasterPage
  Implements DES.ICustomizeDataBoundControl

  Public Sub CustomizeDataBoundControl(ByVal pSender As Object, _
       ByVal pArgs As EventArgs) _
       Implements DES.ICustomizeDataBoundControl.CustomizeDataBoundControl

    Dim vDetailsView As DetailsView = CType(pSender,DetailsView)
    If (Not pSender Is Nothing) Then
      AddHandler vDetailsView.ItemCreated, AddressOf Me.OnDetailsViewItemCreated
    End If

  End Sub' CustomizeDataBoundControl

  Protected Overridable Sub OnDetailsViewItemCreated(ByVal pSender As Object,
       ByVal pArgs As EventArgs)

    Dim vDetailsView As DetailsView = CType(pSender,DetailsView)
    Dim vTEFieldTemplate As DES.TextEditFTUC = _
    CType(DES.Util.GetFieldTemplate(vDetailsView, "Customers", "ContactName"), _
      DES.TextEditFTUC)
    If (Not vTEFieldTemplate Is Nothing) Then
      vTEFieldTemplate.TextBox.BackColor = System.Drawing.Color.Azure
    End If

  End Sub
End Class ' class Site
```

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

Util.GetTableName method

GetTableName is a static/shared support method found on the `PeterBlum.DES.DynamicData.Util` class that can be called within the ItemCreated or RowCreated events of a DataBound control.

Gets the current table name of the DataBound control.

Note: This is also an "extension method". When you declare the namespace PeterBlum.DES.DynamicData, any DataBound control object will include this method. Just omit the first parameter. For example: GridView1.GetTableName() instead of Util.GetTableName(GridView1). Limitation: if you declare the namespace as "DES=PeterBlum.DES.DynamicData", extension methods will not be discovered by the .net framework.

[C#]

```
public static string GetTableName(DataBoundControl pDataBoundControl)
```

[VB]

```
Public Static Function GetTableName(ByVal pDataBoundControl As DataBoundControl)
```

Parameters

pDataBoundControl

A reference to the DataBound control.

Returns

The table name or "" if the DataBound control is not connected to a Dynamic Data table.

Example

The ItemCreated event handler on a DetailsView. The event's method will edit the Field Template associated with the ContactName column of the Customers table. It uses both `Util.GetTableName()` and `Util.GetFieldTemplate()` methods.

[C#]

```
protected virtual void OnDetailsViewItemCreated(object pSender, EventArgs pArgs)
{
  DetailsView vDetailsView = pSender as DetailsView;
  switch (DES.Util.GetTableName(vDetailsView))
  {
    case "Customers":
      DES.TextEditFTUC vTEFieldTemplate =
        DES.Util.GetFieldTemplate(vDetailsView, "ContactName") as DES.TextEditFTUC;
      if (vTEFieldTemplate != null)
        vTEFieldTemplate.TextBox.BackColor = System.Drawing.Color.Azure;
      break;
  } // switch
} // OnDetailsViewItemCreated
```

[VB]

```
Protected Overridable Sub OnDetailsViewItemCreated(ByVal pSender As Object,
    ByVal pArgs As EventArgs)
  Dim vDetailsView As DetailsView = CType(pSender,DetailsView)
  Select Case DES.Util.GetTableName(vDetailsView)
    Case "Customers"
      Dim vTEFieldTemplate As DES.TextEditFTUC = _
        CType(DES.Util.GetFieldTemplate(vDetailsView, "ContactName"), _
          DES.TextEditFTUC)
      If (Not vTEFieldTemplate Is Nothing) Then
        vTEFieldTemplate.TextBox.BackColor = System.Drawing.Color.Azure
      End If
  End Select
End Sub
```

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

Util.GetFieldTemplate method

GetFieldTemplate is a static/shared support methods found on the `PeterBlum.DES.DynamicData.Util` class that can be called within the ItemCreated or RowCreated event of a DataBound control.

Gets the Field Template identified by the Column name and optionally Table name. If you do not specify a table name and your column name appears in multiple tables, it is possible that it will return the wrong Field Template. So always supply the table name when the column name appears in multiple tables.

These methods are always passed a reference to the DataBound control. If you are using any list/grid style control, you also must pass the current row as the second parameter. That row is actually the container of the Field Template. When using a single record style control (FormsView, DetailsView), that second parameter can be omitted or `null`/`Nothing`.

Note: This is also an "extension method". When you declare the namespace PeterBlum.DES.DynamicData, any DataBound control object will include this method. Just omit the first parameter. For example: DetailsView1.GetFieldTemplate("ProductName") instead of Util.GetTableName(DetailsView1, "ProductName"). Limitation: if you declare the namespace as "DES=PeterBlum.DES.DynamicData", extension methods will not be discovered by the .net framework.

[C#]

```
public static FieldTemplateUserControl GetFieldTemplate(
   DataBoundControl pDataBoundControl, Control pContainingRow,
   string pTableName, string pColumnName)
```

```
public static FieldTemplateUserControl GetFieldTemplate(
   DataBoundControl pDataBoundControl,
   string pTableName, string pColumnName)
```

```
public static FieldTemplateUserControl GetFieldTemplate(
   DataBoundControl pDataBoundControl, Control pContainingRow,
   string pColumnName)
```

```
public static FieldTemplateUserControl GetFieldTemplate(
   DataBoundControl pDataBoundControl,
   string pColumnName)
```

[VB]

```
Public Shared Function GetFieldTemplate( _
   ByVal pDataBoundControl As DataBoundControl, ByVal pContainingRow As Control, _
   ByVal pTableName As String, ByVal pColumnName As String) _
      As FieldTemplateUserControl
```

```
Public Shared Function GetFieldTemplate( _
   ByVal pDataBoundControl As DataBoundControl, _
   ByVal pTableName As String, ByVal pColumnName As String) _
      As FieldTemplateUserControl
```

```
Public Shared Function GetFieldTemplate( _
   ByVal pDataBoundControl As DataBoundControl, ByVal pContainingRow As Control, _
   ByVal pColumnName As String) _
      As FieldTemplateUserControl
```

```
Public Shared Function GetFieldTemplate( _
   ByVal pDataBoundControl As DataBoundControl, _
   ByVal pColumnName As String) _
      As FieldTemplateUserControl
```

Parameters

pDataBoundControl

A reference to the DataBound control.

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

pContainingRow

Optional. When using a list-style DataBound control like a GridView or ListView, this is a reference to the containing control for the current row that is being processed within the ItemCreated or RowCreated event.

pTableName

Optional. The name of the table containing the desired column. When specified, it confirms that the same table is used on this DataBound control. Suggested when you have the same column name in several tables.

pColumnName

The column name whose Field Template is requested.

Returns

The FieldTemplateUserControl object or `null`/`Nothing` if the column has no Field Template defined.

Example

Sets up the ItemCreated event handler on a DetailsView. The event's method will edit the Field Template associated with the ContactName column of the Customers table.

[C#]

```
protected virtual void OnDetailsViewItemCreated(object pSender, EventArgs pArgs)
{
  DetailsView vDetailsView = pSender as DetailsView;
  DES.TextEditFTUC vTEFieldTemplate =
    DES.Util.GetFieldTemplate(vDetailsView, "Customers", "ContactName")
        as DES.TextEditFTUC;
  if (vTEFieldTemplate != null)
    vTEFieldTemplate.TextBox.BackColor = System.Drawing.Color.Azure;
} // OnDetailsViewItemCreated
```

[VB]

```
Protected Overridable Sub OnDetailsViewItemCreated(ByVal pSender As Object,
    ByVal pArgs As EventArgs)

  Dim vDetailsView As DetailsView = CType(pSender,DetailsView)
  Dim vTEFieldTemplate As DES.TextEditFTUC = _
    CType(DES.Util.GetFieldTemplate(vDetailsView, "Customers", "ContactName"), _
      DES.TextEditFTUC)
  If (Not vTEFieldTemplate Is Nothing) Then
    vTEFieldTemplate.TextBox.BackColor = System.Drawing.Color.Azure
  End If

End Sub
```

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

Util.GetTable method

GetTable is a static/shared support method found on the `PeterBlum.DES.DynamicData.Util` class that can be called within the ItemCreated or RowCreated event of a DataBound control.

Gets the current MetaTable object from the DataBound control.

Note: This is also an "extension method". When you declare the namespace PeterBlum.DES.DynamicData, any DataBound control object will include this method. Just omit the first parameter. For example: GridView1.GetTable() instead of Util.GetTable(GridView1). Limitation: if you declare the namespace as "DES=PeterBlum.DES.DynamicData", extension methods will not be discovered by the .net framework.

[C#]

```
public static MetaTable GetTable(DataBoundControl pDataBoundControl)
```

[VB]

```
Public Static Function GetTable(ByVal pDataBoundControl As DataBoundControl) _
  As MetaTable
```

Parameters

pDataBoundControl

A reference to the DataBound control.

Returns

The MetaTable object or `null`/`Nothing` if the DataBound control is not connected to a Dynamic Data table.

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

Building Applications without PageTemplates

Many web applications embed the features of ASP.NET Dynamic Data into existing pages or stand-alone pages that do not fit the Page Templates model. When you use Page Templates, you must work within its framework: using the ASP.NET URL Routing system to determine which web form is loaded and accepting that the Page Template is a complete web form. As a result, they do not fit into existing pages.

When working without Page Templates, you still have all of these Dynamic Data features available:

- DataFieldInPattern, DynamicControl, and DynamicField classes to pick the right Field Template for the given data field.
- Automatic scaffolding
- DynamicListView, DynamicFormView, and DynamicPattern control. (You can use other DataBound controls too.)
- DynamicDataManager Control and Adapters
- Dynamic Data enabled DataSource class like LinqDataSource and EntityDataSource
- DES's Customizer Control

This section shows how to add these controls to your web form.

Click on any of these topics to jump to them:

- Steps for adding Dynamic Data Features into your forms
  - Using DES.DynamicControl
  - Using DES.DynamicField
  - Using the DataFieldInPattern class
  - Using the DataFieldListInPattern Class
  - Using the NoDataFieldsInPattern class
  - Using the ScaffoldInPattern class
  - Using TemplateNotInPattern
- Pattern Templates: Reusable formatting for DynamicControls
  - The DynamicPattern Control
  - The DynamicListView Control
  - The DynamicFormView Control
  - The DynamicStylesTag Control
  - The NamedStyle Class
  - The DynamicPartsPatterns Control
- DynamicDataManager Control and Adapters
- DES.DynamicControl: The heart of Dynamic Data
- DES.DynamicField: Using Dynamic Data in GridView and DetailsView
- Customizer Control

Sample files for all types of DataBound controls can be found in these folders:

[DES Product Folder]/DynamicData/*language*/Samples/

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

Steps for adding Dynamic Data Features into your forms

Use these steps to add any type of DataBound control, whether list or single record.

Big Picture

- Add a DataBound control. For a list, the suggested control is a DynamicListView. For a single record, the suggested control is a DynamicFormView. Other choices: GridView, ListView, DetailsView and FormView
- Add a DataSource that is supported by Dynamic Data, such as LinqDataSource or EntityDataSource. Connect your DataBound control to it.
- Add a DynamicDataManager, Customizer, and ValidationSummary control.
- Add a DataBound Control Adapter object to the **Adapters** property of the DynamicDataManager. Set its **DataBoundControlID**, **ValidationSummaryControlID**, and **CustomizerControlID** properties to the controls already added. Set its **SupportsEditMode** and **SupportsInsertMode** properties, and other properties.
- Declare the data fields shown within the DataBound control. You can optionally use Automatic Scaffolding.
- Decorate the form with Filter fields and paging capabilities. Consider the DynamicControlPanel control.

Detailed steps

These steps ask you 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. Declare the following namespaces in your code behind file:

   [C#]

   ```
   using System.Web.DynamicData;
   using DES=PeterBlum.DES.DynamicData;
   ```

   [VB]

   ```
   Imports System.Web.DynamicData
   Imports DES=PeterBlum.DES.DynamicData
   ```

   Note: Use "DES" in front of any type found in the PeterBlum.DES.DynamicData namespace within the code on this page. For example, DES.DynamicDataManager. (If you use DynamicDataManager, it will use the one in System.Web.DynamicData.)

2. Add the DES DynamicDataManager control. Its position does not matter since it does not output any HTML.

   ```
   <desDD:DynamicDataManager ID="DynamicDataManager1" runat="server" >
   </desDD:DynamicDataManager>
   ```

   Set its **AutoLoadForeignKeys** property to `true` when using LINQ to SQL and you want scaffolding to display data from foreign keys found in your table. (Not used with the Entity Framework.)

3. Add the DES Customizer Control. Its position does not matter since it does not output any HTML.

   ```
   <desDD:Customizer ID="Customizer1" runat="server" >
   </desDD:Customizer>
   ```

4. Add the DES ValidationSummary control that will show the errors for the DataBound control. It can be the same ValidationSummary used for other controls on the page. You will not have to change its **Group** property to reflect different Validation Groups. That's handled automatically.

   When using this control within an UpdatePanel, be sure to set its **InAJAXUpdate** property to `true`.

   ```
   <des:ValidationSummary ID="ValidationSummary1" runat="server"
      InAJAXUpdate="True" />
   ```

CONTINUED ON THE NEXT PAGE

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

5. Add the DataSource control such as the LinqDataSource or EntityDataSource.

   If you use another DataSource, it must implement `System.Web.DynamicData.IDynamicDataSource`.

6. Assign these properties on the DataSource control:

   - **ContextTypeName** – The class name of the Data Context class.
   - **TableName** (LinqDataSource only) – The name of the table to view. This is the actual table name, not the name of an EntityTable class or value determined by a DisplayNameAttribute.
   - **EntitySetName** (EntityDataSource only) – The name of the table to view. This is the actual table name, not the name of an EntityTable class or value determined by a DisplayNameAttribute.
   - **EnableUpdate** – Set to `true` if you allow Edit mode. It defaults to `false`.
   - **EnableInsert** – Set to `true` if you allow Insert mode. It defaults to `false`.
   - **EnableDelete** – Set to `true` if you allow deletion. It defaults to `false`.

```
<asp:LinqDataSource ID="RecordDetailsDataSource" runat="server"
   ContextTypeName="NorthWindDataContext" TableName="Products"
   EnableUpdate="true" EnableInsert="true" >
</asp:LinqDataSource>
```

```
<asp:EntityDataSource ID="RecordDetailsDataSource" runat="server"
   ContextTypeName="NorthWindDataContext" EntitySetName="Products"
   EnableUpdate="true" EnableInsert="true" >
</asp:EntityDataSource>
```

7. Add the DataBound control. Your choices are:

   - DynamicListView – To show multiple records where you use Pattern Templates to control the format. (It mimics the GridView by default.) It applies automatic scaffolding when its ItemsInPattern collection is empty or you set **AutoScaffoldDataFields** to `True`.

     This is the recommended control for lists.

   - GridView – To show multiple records in a table format. Supports automatic scaffolding when **AutoGenerateColumns** is `true`.
   - ListView – To show multiple records where you determine the format through HTML and web controls. Does not support automatic scaffolding unless you use Pattern Templates with its ScaffoldColumnsInPattern object.
   - DynamicFormView – To show a single record where you use Pattern Templates to control the format. (It mimics the DetailsView by default.) It applies automatic scaffolding when its ItemsInPattern collection is empty or you set **AutoScaffoldDataFields** to `True`.

     This is the recommended control for a single record view.

   - DetailsView – To show a single record in a table format with two columns: colum name and column value. Supports automatic scaffolding when **AutoGenerateRows** is `true`.
   - FormView – To show a single record where you determine the format through HTML and web controls. Does not support automatic scaffolding unless you use Pattern Templates with its ScaffoldColumnsInPattern object.

   You will enhance your DataBound control in later steps.

8. Set the **DataSourceID** property of the DataBound control to the ID of the DataSource control.

```
<asp:DynamicFormView ID="RecordDetails" runat="server"
   DataSourceID="RecordDetailsDataSource" >
</asp:DynamicFormView>
```

CONTINUED ON THE NEXT PAGE

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

9. Connect the DataBound Control Adapters to the DataBound control and ValidationSummary. You will add the following objects to the **Adapters** property of the DES DynamicDataManager. See "DynamicDataManager Control and Adapters".

DataBound control	Adapter class
DynamicListView	DynamicListViewAdapter
GridView	GridViewAdapter
ListView	ListViewAdapter
DynamicFormView	DynamicFormViewAdapter
DetailsView	DetailsViewAdapter
FormView	FormViewAdapter

When you have multiple DataBound control on the page, add one DataBound Control Adapter for each.

These classes have many properties to consider. See "The BaseDataBoundControlAdapter class".

Generally you set **DataBoundControlID** and **ValidationSummaryControlID**. When the DataBound control is not using Edit mode, set **SupportsEditMode** to `false`. When its not using Insert mode, set **SupportsInsertMode** to `false`.

Example: DynamicFormView in Edit.aspx

```
<desDD:DynamicDataManager ID="DynamicDataManager1" runat="server"
   AutoLoadForeignKeys="true" >
   <Adapters>
      <desDD:DynamicFormViewAdapter DataBoundControlID="RecordDetails"
         ValidationSummaryControlID="ValidationSummary1"
         SupportsEditMode="True" SupportsInsertMode="False"
         CommandsUseRouting="true"
         PostUpdateAction="List" PostCancelAction="List" />
   </Adapters>
</desDD:DynamicDataManager>
```

Example: DynamicListView and DynamicFormView in ListDetails.aspx

In this example, assume that there is one ValidationSummary control shared between the two DataBound controls.

```
<desDD:DynamicDataManager ID="DynamicDataManager1" runat="server"
   AutoLoadForeignKeys="true" >
   <Adapters>
      <desDD:DynamicListViewAdapter DataBoundControlID="RecordsList"
         ConnectedToControlID="RecordDetails"
         ValidationSummaryControlID="ValidationSummary1"
         SupportsInsertMode="false" SupportsEditMode="true"
         CommandsUseRouting="true" />
      <desDD:DynamicFormViewAdapter DataBoundControlID="RecordDetails"
         ConnectedToControlID="RecordsList"
         ValidationSummaryControlID="ValidationSummary1"
         SupportsInsertMode="true" SupportsEditMode="true"
         CommandsUseRouting="true" />
   </Adapter>
</desDD:DynamicDataManager>
```

CONTINUED ON THE NEXT PAGE

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

10. Establish any event handlers on the DataBound control. Use the sample files for guidance. You do not need any event handlers to assist DES Dynamic Data. It establishes them when you register the control with the DataBound Control Adapter in your DynamicDataManager control.
    - When you change the Mode on DataBound control, also call the `DynamicDataManager.ChangeMode()` method passing a value of the enumerated type `System.Web.UI.WebControls.DataBoundControlMode`: `ReadOnly`, `Insert`, or `Edit`.
    - Almost all event handlers that were defined in the native Page Templates are internally handled within the DataBound Control Adapter.
    - Generally the event handlers you need provide interaction between a DataBound control and non-DataBound control. Examples: Filters, DynamicControlPanel, and Insert button. Look at the Page Templates and **[DES Product Folder]/DynamicData/[language]/Samples** folders for examples.
11. Consider adding paging, especially to list and readonly detail views. DES includes a powerful paging control called DynamicControlPanel. See "The DynamicControlPanel Control". Many controls have more limited pagers within their properties. Here are some options:

DataBound control	Available Paging tools
DynamicListView	DynamicControlPanel control above and/or below the control. Add the DynamicControlPanel into the `<HeaderTemplate>` and/or `<FooterTemplate>` of the Pattern Template supporting this control.
GridView	DynamicControlPanel control above and/or below the control. `<PagerSettings>` property or the `<PagerTemplate>` property with the DynamicControlPanel or the GridViewPager user control for paging within a GridView in **[web application]/DynamicData/Content**.
ListView	DynamicControlPanel control above and/or below the control. `<LayoutTemplate>` property. Consider using the DynamicControlPanel
DynamicFormView	DynamicControlPanel control above and/or below the control. Add the DynamicControlPanel into the `<HeaderTemplate>` and/or `<FooterTemplate>` of the Pattern Template supporting this control.
DetailsView	DynamicControlPanel control above and/or below the control. `<PagerSettings>` property or the `<PagerTemplate>` property with the DynamicControlPanel or your own code
FormView	DynamicControlPanel control above and/or below the control. `<PagerSettings>` property or the `<PagerTemplate>` property with the DynamicControlPanel or your own code

12. If you want to offer filtering controls to a list style interface, consider the FilterRepeater. Additional resources:
    - Extending the FilterRepeater – by Marcin Dobosz.
    - AdvancedFilterRepeater in Dynamic Data Futures.

At this point, you have done everything except establish the actual page output from the DataBound controls. You may want to save a copy of your web form for later use when creating another web form with DES Dynamic Data.

CONTINUED ON THE NEXT PAGE

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

13. Set up the list of data fields shown within the DataBound control.

    Sample files for can be found in these folders:

    [C#]

    [DES Product Folder]/DynamicData/CS/Samples/

    [VB]

    [DES Product Folder]/DynamicData/VB/Samples/

Using Automatic Scaffolding to generate the list of data fields

See "Automatic Scaffolding Overview" and "Customizing Automatic Scaffolding".

DataBound Control	How to enable scaffolding
DynamicListView	It applies automatic scaffolding when its **ItemsInPattern** collection is empty or you set **AutoScaffoldDataFields** to `True`.
GridView	Set **AutoGenerateColumns** property to `true`
ListView	Add a DynamicPattern control to the various itemtemplates. Assign the **PatternTemplateName** property on DynamicPattern to **GridView.ascx**, **ListOfDetailViews.ascx**, or a customized version of those Pattern Templates. Set the **Mode** property to "ReadOnly", "Edit" or "Insert" depending on the type of ItemTemplate. It applies automatic scaffolding when its **ItemsInPattern** collection is empty or you set **AutoScaffoldDataFields** to `True`.
DynamicFormView	It applies automatic scaffolding when its **ItemsInPattern** collection is empty or you set **AutoScaffoldDataFields** to `True`.
DetailsView	Set **AutoGenerateRows** property to `true`
FormView	Add a DynamicPattern control to the various itemtemplates. Assign the **PatternTemplateName** property on DynamicPattern to **DetailsView.ascx** or a customized version of that Pattern Template. Set the **Mode** property to "ReadOnly", "Edit" or "Insert" depending on the type of ItemTemplate. It applies automatic scaffolding when its **ItemsInPattern** collection is empty or you set **AutoScaffoldDataFields** to `True`.

CONTINUED ON THE NEXT PAGE

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

Specifying the list of data fields in DynamicListView, GridView and ListView controls

DataBound Control	How to add a field

DynamicListView	Add DataFieldInPattern objects to the **Fields** property of the DynamicListView control.
	`<ItemsInPattern>`
	`  <desDD:DataFieldInPattern DataField="ProductName" />`
	`  <desDD:DataFieldInPattern DataField="Price" UIHint="Price" />`
	`</ItemsInPattern>`

GridView	Add the `PeterBlum.DES.DynamicData.DynamicField` object to the **Columns** property of the GridView control.
	`<Columns>`
	`  <desDD:DynamicField DataField="ProductName" />`
	`  <desDD:DynamicField DataField="Price" UIHint="Price" />`
	`</Columns>`

ListView with Pattern Templates	Add a DynamicPattern control to the various itemtemplates.
	Assign the **PatternTemplateName** property on DynamicPattern to "GridView", "ListView" or a customized version of those Pattern Templates.
	Set the **Mode** property to "ReadOnly", "Edit" or "Insert" depending on the type of ItemTemplate.
	Add DataFieldInPattern objects to the **Fields** property on the DynamicPattern control.
	`<ItemTemplate>`
	`  <desDD:DynamicPattern id="ReadOnlyFields" runat="server"`
	`  PatternTemplateName="GridView" Mode="ReadOnly" >`
	`  <ItemsInPattern>`
	`  <desDD:DataFieldInPattern DataField="ProductName" />`
	`  <desDD:DataFieldInPattern DataField="Price" UIHint="Price" />`
	`  </ItemsInPattern>`
	`  </desDD:DynamicPattern>`
	`</ItemTemplate>`

ListView without Pattern Templates	Add DynamicControls, DynamicLabels, HTML and other web controls to the various templates. • Use the DynamicControl where you want Dynamic Data to setup a Data Field. • Use the DynamicLabel control where you want a label with the column name that gets that name using metadata.

CONTINUED ON THE NEXT PAGE

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

Specifying the list of data fields in DynamicFormView, DetailsView and FormView controls

DataBound Control	How to add a field

DynamicFormView	Add DataFieldInPattern objects to the **Fields** property of the DynamicFormView control.
	`<ItemsInPattern>`
	`  <desDD:DataFieldInPattern DataField="ProductName" />`
	`  <desDD:DataFieldInPattern DataField="Price" UIHint="Price" />`
	`</ItemsInPattern>`

DetailsView	Add the `PeterBlum.DES.DynamicData.DynamicField` object to the **Rows** property of the DetailsView control.
	`<Fields>`
	`  <desDD:DynamicField DataField="ProductName" />`
	`  <desDD:DynamicField DataField="Price" UIHint="Price" />`
	`</Fields>`

FormView with Pattern Templates	Add a DynamicPattern control to the various itemtemplates.
	Assign the **PatternTemplateName** property on DynamicPartsPatterns to "DetailsView" or a customized version of that Pattern Template.
	Set the **Mode** property to "ReadOnly", "Edit" or "Insert" depending on the type of ItemTemplate.
	Add DataFieldInPattern objects to the **Fields** property on the DynamicPattern control.
	`<ItemTemplate>`
	`  <desDD:DynamicPattern id="ReadOnlyFields" runat="server"`
	`  PatternTemplateName="DetailsView" Mode="ReadOnly" >`
	`  <ItemsInPattern>`
	`  <desDD:DataFieldInPattern DataField="ProductName" />`
	`  <desDD:DataFieldInPattern DataField="Price" UIHint="Price" />`
	`  </ItemsInPattern>`
	`  </desDD:DynamicPattern>`
	`</ItemTemplate>`

FormView without Pattern Templates	Add DynamicControls, DynamicLabels, HTML and other web controls to the various templates. • Use the DynamicControl where you want DynamicData to setup a Data Field. • Use the DynamicLabel control where you want a label with the column name that gets that name using metadata.

CONTINUED ON THE NEXT PAGE

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

14. Add buttons to handle the desired commands: Edit, Select, Delete, Details, Update (save in Edit mode), Insert (save in Insert mode), and Cancel.

Using the DataBound control's ability to generate the buttons automatically

DataBound Control	How to show the buttons
DynamicListView	**AutoGenerateButtons** = `true` The appearance is determined by Pattern Templates. There are two predefined, "Buttons" and "LinkButtons". "LinkButtons" is the default. Change the type in the **ButtonsNamedPart** property. Further customization can be done within the Pattern Template files ("ButtonsForList" or "LinkButtonsForList").
GridView	See "Using Properties on the GridView and DetailsView".

ListView	Add a DynamicPattern control to the various itemtemplates where you want the buttons.
	Assign the **PatternTemplateName** property on DynamicPattern to "ButtonsForList", "LinkButtonsForList" or a customized version of those Pattern Templates.
	Set the **Mode** property to "ReadOnly", "Edit" or "Insert" depending on the type of ItemTemplate.
	`<ItemTemplate>`
	`  <desDD:DynamicPattern id="ReadOnlyFields" runat="server"`
	`  PatternTemplateName="ButtonsForList" Mode="ReadOnly" >`
	`  </desDD:DynamicPattern>`
	`</ItemTemplate>`

DynamicFormView	**AutoGenerateButtons** = `true` The appearance is determined by Pattern Templates. There are two predefined, "Buttons" and "LinkButtons". "LinkButtons" is the default. Change the type in the **ButtonsNamedPart** property. Further customization can be done within the Pattern Template files ("DetailsViewButtons" or "DetailsViewLinkButtons").
DetailsView	See "Using Properties on the GridView and DetailsView".

FormView	Add a DynamicPattern control to the various itemtemplates where you want the buttons.
	Assign the **PatternTemplateName** property on DynamicPattern to "DetailsViewButtons" or "DetailsViewLinkButtons" or a customized version of those Pattern Templates.
	Set the **Mode** property to "ReadOnly", "Edit" or "Insert" depending on the type of ItemTemplate.
	`<ItemTemplate>`
	`  <desDD:DynamicPattern id="ReadOnlyFields" runat="server"`
	`  PatternTemplateName="DetailsViewButtons" Mode="ReadOnly" >`
	`  </desDD:DynamicPattern>`
	`</ItemTemplate>`

CONTINUED ON THE NEXT PAGE

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

Declaring the buttons within the list of fields

Sometimes the positioning or features of the automatic button generation don't fit your needs. You can declare the buttons more explicitly within the list of fields declared for the DataBound control. For instance, if you want buttons to appear before your data fields, do the following prior to the list of data fields.

With this approach, you can set properties, such as the text of the buttons, or their visibility.

DataBound Control	How to show the buttons

DynamicListView	**AutoGenerateButtons** = `None`
	Within the **ItemsInPattern** property, add a ButtonsInPattern object where you want the buttons.
	Assign the **NamedPart** property to "Buttons" or "LinkButtons".
	The Pattern Templates have numerous properties. See "Passing properties to be used by the PatternTemplate's own code". They can be declared as properties on the ButtonsInPattern object, such as the **ShowSelectButton** property shown here.
	`<ItemsInPattern>`
	`  <desDD:ButtonsInPattern NamedPart="Buttons"`
	`  ShowSelectButton="True"/>`
	`</ItemsInPattern>`

GridView	Add the CommandField to the **Columns** property. In that case, set its **ShowEditButton**, **ShowInsertButton**, **ShowDeleteButton**, **ShowSelectButton**, and **ShowCancelButton** properties to `false`. Alternatively, use the ButtonField class, defining the **CommandName** property.

ListView	Add a DynamicPattern control to the various itemtemplates where you want the buttons.
	Assign the **PatternTemplateName** property on DynamicPattern to "ButtonsForList", "LinkButtonsForList" or a customized version of those Pattern Templates.
	Set the **Mode** property to "ReadOnly", "Edit" or "Insert" depending on the type of ItemTemplate.
	`<ItemTemplate>`
	`  <desDD:DynamicPattern id="ReadOnlyFields" runat="server"`
	`  PatternTemplateName="ButtonsForList" Mode="ReadOnly" >`
	`  </desDD:DynamicPattern>`
	`</ItemTemplate>`

DynamicFormView	Same as DynamicListView, above.
DetailsView	Add the CommandField to the **Rows** property. In that case, set its **ShowEditButton**, **ShowInsertButton**, **ShowDeleteButton**, **ShowSelectButton**, and **ShowCancelButton** properties to `false`. Alternatively, use the ButtonField class, defining the **CommandName** property.

FormView	Add a DynamicPattern control to the various itemtemplates where you want the buttons.
	Assign the **PatternTemplateName** property on DynamicPattern to "DetailsViewButtons" or "DetailsViewLinkButtons" or a customized version of those Pattern Templates.
	Set the **Mode** property to "ReadOnly", "Edit" or "Insert" depending on the type of ItemTemplate.
	`<ItemTemplate>`
	`  <desDD:DynamicPattern id="ReadOnlyFields" runat="server"`
	`  PatternTemplateName="DetailsViewButtons" Mode="ReadOnly" >`
	`  </desDD:DynamicPattern>`
	`</ItemTemplate>`

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

15. If you need to programmatically customize the attributes for a column, use the IUICustomizeColumn interface. For example, each Field Template gets a description for a tooltip from the DescriptionAttribute. You can define or replace the description within the `CustomizeColumn()` method by setting it on the **Description** property of the *pModifiableColumn* property.
16. The DynamicField and DynamicControl objects let you specify values for public properties on the Field Template it uses. If you need to do the same programmatically, use the ICustomizeFieldTemplate interface.

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

Pattern Templates: Reusable formatting for DynamicControls

Pattern Templates are perhaps the most important tool to creating web forms. They are User Controls that contain a block of HTML and various controls – including DynamicControls – that describe a pattern. Your ListView and FormView controls require much less setup because instead of writing the HTML and controls of a pattern, you include a Pattern Template by specifying its name in a **DynamicPattern** control.

Click on any of these topics to jump to them:

- Overview
- The DynamicPattern Control
- The DataFieldInPattern Class
- The DataFieldListInPattern Class
- The ScaffoldInPattern Class
- The NoDataFieldsInPattern Class
- The TemplateNotInPattern Class
- The NamedStyle Class

- The DynamicListView Control
- The DynamicFormView Control

- Using the PatternTemplates shipped with DES Dynamic Data

- Customizing and creating your own PatternTemplates
- The DynamicStylesTag Control
- The DynamicPartsPatterns Control

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

Overview

A Pattern Template file contains a block of HTML and web controls, including DynamicControls and DynamicLabels, that you will reuse. It's a smart UserControl that knows how to assign the **DataField** property on DynamicControls and DynamicLabels to data field names the page developer passes in. The page developer uses the DynamicPattern, DynamicListView, or DynamicFormView control to identify the Pattern Template file name and its list of data field names.

A Pattern Template's formatting can also be customized by the page developer, who supplies "Named Styles" and property settings that have been declared on the individual Pattern Template.

DES ships with several Pattern Templates including two that mimic the appearance of the GridView and DetailsView controls.

Click on any of these topics to jump to them:

- Let's start with an example
  - Switching to a PatternTemplate: The DynamicPattern control
- Named Styles: Your pages control the formatting
- The Powerful DynamicListView and DynamicFormView controls
- Location of your PatternTemplate files
- PatternTemplates shipped with DES Dynamic Data
  - Style sheets

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

Let's start with an example

One example of a pattern is the two column row where the first column is the field label and the second column is the data. Here is an example within the `<ItemTemplate>` tag of a FormView where a Pattern Template can assist you. It has 3 rows, each with the same HTML and controls except for the name in the **DataField** property.

```
<ItemTemplate>
<tr class="RowStyle">
  <td class="LabelColumn">
    <desDD:DynamicLabel id="DynamicLabel1" runat="server"
      AssociatedControlID="DynamicControl1" CssClass="LabelControl" />
  </td>
  <td class="DataColumn">
    <desDD:DynamicControl id="DynamicControl1" runat="server"
      DataField="ProductName" />
  </td>
</tr>

<tr class="RowStyle">
  <td class="LabelColumn">
    <desDD:DynamicLabel id="DynamicLabel2" runat="server"
      AssociatedControlID="DynamicControl2" CssClass="LabelControl" />
  </td>
  <td class="DataColumn">
    <desDD:DynamicControl id="DynamicControl2" runat="server"
      DataField="Description" />
  </td>
</tr>

<tr class="RowStyle">
  <td class="LabelColumn">
    <desDD:DynamicLabel id="DynamicLabel3" runat="server"
      AssociatedControlID="DynamicControl3" CssClass="LabelControl" />
  </td>
  <td class="DataColumn">
    <desDD:DynamicControl id="DynamicControl3" runat="server"
      DataField="Price" />
  </td>
</tr>

</ItemTemplate>
```

Here is a Pattern Template for a two column row from the above example. Note two things. 1) It inherits from the class `PeterBlum.DES.DynamicData.DataFieldsPatternTemplate` (there are several available classes from which to subclass). 2) The DynamicControl lacks the **DataField** property.

```
<%@ Control Language="C#"
  Inherits="PeterBlum.DES.DynamicData.DataFieldsPatternTemplate" %>
<tr class="RowStyle">
  <td class="LabelColumn">
    <desDD:DynamicLabel id="DynamicLabel1" runat="server"
      AssociatedControlID="DynamicControl1" CssClass="LabelControl" />
  </td>
  <td class="DataColumn">
    <desDD:DynamicControl id="DynamicControl1" runat="server" />
  </td>
</tr>
```

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

Switching to a PatternTemplate: The DynamicPattern control

To use a Pattern Template, you add a **DynamicPattern** control into your ListView or FormView controls. The DynamicPattern control identifies the Pattern Template file name and a list of data field names which to bind to the DynamicControls and DynamicLabels found in the Pattern Template.

Here is an example of DynamicPattern web controls within the `<ItemTemplate>` of a FormView control that uses the above Pattern Template, which for this example, is given the file name "TwoColumns.ascx".

```
<ItemTemplate>
  <desDD:DynamicPattern id="DynamicPattern1" runat="server"
    PatternTemplateName="TwoColumns" DataField="ProductName" />
  <desDD:DynamicPattern id="DynamicPattern2" runat="server"
    PatternTemplateName="TwoColumns" DataField="Description" />
  <desDD:DynamicPattern id="DynamicPattern3" runat="server"
    PatternTemplateName="TwoColumns" DataField="Price" />
</ItemTemplate>
```

For more on the DynamicPattern control, see "The DynamicPattern Control".

Note: The v4.0 release of ASP.NET Dynamic Data introduces "Entity Templates". That is a very simplistic implementation of Pattern Templates. While you can use Entity Templates within DES Dynamic Data, you will find Pattern Templates a better toolset.

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

Named Styles: Your pages control the formatting

Since Pattern Templates need to be used in many different situations, they should not force you to use a specific style sheet, like the above example shows. In addition, due to the layout of the page, they may need adjustments to their width and perhaps other elements. Pattern Templates let you move the definition of these things into the page that uses them.

Click on any of these topics to jump to them:

- The NamedStyle class
- The DynamicStylesTag control
- Other ways to apply Named Styles
- Merging Style Sheet Classes

The NamedStyle class

Pattern Templates use "**Named styles**" to modify the class=, style=, and other HTML attributes within its web controls and HTML. The class `PeterBlum.DES.DynamicData.NamedStyle` class contains class=, style=, and other HTML attributes that will be applied to the Pattern Template, replacing any matching attribute.

The Pattern Template defines a list of names to NamedStyles that it supports. You create a NamedStyle object within the NamedStyles property of a DynamicPattern, DynamicListView, or DynamicFormView control and assign it's **Name** property to one of the names supported by the Pattern Template. Also assign the alternative style sheet class to its **CssClass** property and element styles to its **Style** property. The Pattern Template will find your NamedStyle and merges your values with its own. (When both use the same value, the NamedStyle object takes precidence).

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
     PatternTemplateName="TwoColumns" DataField="ProductName" >
   <NamedStyles>
      <desDD:NamedStyle Name="Row" CssClass="CustomRowStyle" />
      <desDD:NamedStyle Name="LabelColumn" Style="width:150px;" />
   </NamedStyles>
</desDD:DynamicPattern>
```

You can narrow down the matching by specifying a data field in the **DataFields** property, a Category name in the **Categories** property, and user security roles in the **Roles** property. Categories are defined with the CategoryAttribute on the metadata for data fields.

In this example, the style sheet "PriceStyle" applies when to the "Price" data field set within the Row NamedStyle. The font is made bold on any data field with the CategoryAttribute set to "Title" within the Row NamedStyle.

```
<desDD:DynamicListView id="DynamicListView1" runat="server" >
   <NamedStyles>
      <desDD:NamedStyle Name="Row" Categories="Title" Style="font-weight:bold;" />
      <desDD:NamedStyle Name="Row" DataFields="Price" CssClass="PriceStyle" />
      <desDD:NamedStyle Name="Row" DataFields="Price" Role="Admin"
         CssClass="PriceStyle Admin" />
   </NamedStyles>
</desDD:DynamicListView>
```

Because a NamedStyle can evaluate data field, category, and user role names, you may specify several NamedStyles with the same name, but different combinations of the other evaluators. It will always use the most specific match over a less specific match. For example, in the above example, it uses the third NamedStyle when "Row" is being generated. If that is not generating output for "Admin", it then uses the second NamedStyle. If two NamedStyles could match and have the same number of evaluators, it uses the order specified in the **NamedStyles** property.

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

The DynamicStylesTag control

Within the Pattern Template, HTML tags using named styles are often replaced by **DynamicStylesTag** controls. Use the DynamicStylesTag control to generate the HTML tag with style and attributes that come from the Named Style, or default values if that Named Style is not specified in the DynamicPattern control.

Here is a Pattern Template example that starts without any Named Styles support.

```
<tr class="RowStyle">
  <td class="LabelColumn" style="white-space:nowrap;">
    <desDD:DynamicLabel id="DynamicLabel1" runat="server"
      AssociatedControlID="DynamicControl1" CssClass="LabelControl" />
  </td>
  <td class="DataColumn" style="white-space:nowrap;">
    <desDD:DynamicControl id="DynamicControl1" runat="server" />
  </td>
</tr>
```

The DynamicStylesTag controls have been applied here. The **Tag** property indicates the HTML tag to create. The **NamedStyle** property is the name to locate amongst the supplied Named Styles. The **CssClass** and Style properties contain the values used if a Named Style does not offer its own value.

```
<desDD:DynamicStylesTag runat="server" Tag="Tr" NamedStyle="Row" CssClass="RowStyle" >

  <desDD:DynamicStylesTag runat="server" Tag="Td" NamedStyle="LabelColumn"
      CssClass="LabelColumn" Style="white-space:nowrap;" >
    <desDD:DynamicLabel ID="DynamicLabel1" runat="server"
        AssociatedControlID="DynamicControl1" CssClass="LabelControl" />
  </desDD:DynamicStylesTag>

  <desDD:DynamicStylesTag runat="server" Tag="Td" NamedStyle="DataColumn"
      CssClass="DataColumn" Style="white-space:nowrap;" >
    <desDD:DynamicControl ID="DynamicControl1" runat="server" />
  </desDD:DynamicStylesTag>

</desDD:DynamicStylesTag>
```

When the DynamicPattern control does not have any NamedStyle objects declared, it generates HTML output that looks like the original `<tr>` and `<td>` tags above.

When you specify these NamedStyles:

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
     PatternTemplateName="TwoColumns" DataField="ProductName" >
  <NamedStyles>
    <desDD:NamedStyle Name="Row" CssClass="CustomRowStyle" />
    <desDD:NamedStyle Name="LabelColumn" CssClass="BoldText"
      Style="width:150px;"  />
  </NamedStyles>
</desDD:DynamicPattern>
```

You get this HTML output:

```
<tr class="CustomRowStyle">
  <td class="BoldText" style="white-space:nowrap; width:150px;">
    <desDD:DynamicLabel id="DynamicLabel1" runat="server"
      AssociatedControlID="DynamicControl1" CssClass="LabelControl" />
  </td>
  <td class="DataColumn" style="white-space:nowrap;">
    <desDD:DynamicControl id="DynamicControl1" runat="server" />
  </td>
</tr>
```

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

Other ways to apply Named Styles

Instead of using the DynamicStylesTag control, you can embed a call to `GetTagAttributes()` into your HTML like this:

```
<tr <% =GetTagAttribute("RowStyle", "Row") %> >
  <td <% =GetTagAttribute("LabelColumn", "LabelColumn", "", "white-space:nowrap;") %> >
    <desDD:DynamicLabel id="DynamicLabel1" runat="server"
      AssociatedControlID="DynamicControl1" CssClass="LabelControl" />
  </td>
  <td <% =GetTagAttribute("DataColumn", "DataColumn", "", "white-space:nowrap;") %> >
    <desDD:DynamicControl id="DynamicControl1" runat="server" />
  </td>
</tr>
```

When creating HTML programmatically within a Pattern Template, call the `ApplyNamedStyle()` method passing your control and the desired Named Style.

```
ApplyNamedStyle("LabelColumn", DynamicLabel1)
```

Merging Style Sheet Classes

Often you need to add styles to the existing style sheet class. For example, you want to merge the style sheet class "LabelColumn" with this class:

```
.W150
{
  width:150px;
}
```

The Named Style class allows merging styles by adding a + character before the style sheet class name like this:

```
<desDD:NamedStyle Name="LabelColumn" CssClass="+W150" />
```

This creates a `class=` attribute on the LabelColumn of `class="LabelColumn W150"`.

The NamedStyle class lets you supply styles to the HTML tag within its **Style** property. When used, they override a style in the style sheet class definition for that HTML tag.

```
<desDD:NamedStyle Name="LabelColumn" Style="min-width:50px;" />
```

For more on Named Styles and the DynamicStylesTag, see "The DynamicStylesTag Control" and "The NamedStyle Class".

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

The Powerful DynamicListView and DynamicFormView controls

The GridView and DetailsView are often used on web forms with Dynamic Data because they are so easy to setup. They have all of the HTML generation rules built in. You merely establish a list of DynamicFields and ButtonFields, or let scaffolding create the DynamicFields and other rules create the buttons for you. In fact, if you drop the basic GridView or DetailsView onto the page and assign a DataSource, they generate HTML for the data.

```
<asp:GridView ID="GridView1" runat="server"
  DataSourceID="DataSource1" AutoGenerateEditButton="true" />

<asp:DetailsView ID="DetailsView1" runat="server"
   DataSourceID="DataSource1" AutoGenerateRows="false" >
   <Fields>
      <asp:DynamicField DataField="ProductName" />
      <asp:DynamicField DataField="Description" />
      <asp:DynamicField DataField="Price" />
      <asp:CommandField />
   </Fields>
</asp:DetailsView>
```

All this easy of setup comes with a significant limitation. You cannot control much of the HTML generated. At times, you will use the TemplateField to replace the contents of row with new HTML and web controls, but this is still restricted by the overall design of the GridView and DetailsView controls.

When users encounter these limitations, they switch to ListView and FormView controls, where they have to build 100% of the HTML and web controls. That is a lot of work especially when you needed to tweak just a few elements of the original control. They also cannot use scaffolding anymore. Users should take advantage of Pattern Templates to reduce the setup, but that is still more work than necessary.

Along come the **DynamicListView** and **DynamicFormView** controls to resolve this difficulty in setup.

The DynamicListView and DynamicFormView controls are versions of ListView and FormView that mimic the easy setup of the GridView and DetailsView, including scaffolding support. They internally use the DynamicPattern control to load Pattern Templates that recreate the HTML of the GridView and DetailsView. With them in place, you can tweak the HTML within a specific Pattern Template while preserving the work done to create the original control.

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
   DataSourceID="DataSource1" AutoGenerateButtons="Last" />

<desDD:DynamicFormView ID="DynamicFormView1" runat="server"
   DataSourceID="DataSource1"
   AutoGenerateButtons="None" >
   <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="ProductName" />
      <desDD:DataFieldInPattern DataField="Description" />
      <desDD:DataFieldInPattern DataField="Price" />
      <desDD:ButtonsInPattern NamedPart="LinkButtons" />
   </ItemsInPattern>
</desDD:DynamicFormView>
```

Note: Normally these controls specify a Pattern Template by its name in their PatternTemplateName property. When not specified, DynamicListView uses "GridView" which minics the GridView control. DynamicFormView uses "DetailsView" which mimics the DetailsView control.

These two controls are the preferred option over GridView, ListView, DetailsView and FormView as they balance easy setup with flexibility.

If these controls simplify things too much, you can still use some of their underlying technology within ListView and FormView control. Basically, you will drop DynamicPattern controls into their template properties, setting up the same list of data field names and Named Styles in each. Doing so disables the automatic code generation for the Template, instead using whatever you placed in the Template property.

Shortcut: Go into design mode of Visual Studio and select the "Manage the auto generation of Templates…" command from the SmartTag.

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

The end result is the DynamicListView and DynamicFormView can be used as anything from a simplistic GridView and DetailsView to a fully configurable ListView and FormView.

To learn more, see “The DynamicListView Control” and “The DynamicFormView Control”.

Location of your PatternTemplate files

Pattern Template files are stored in the **[web app]/DynamicData/PatternTemplates** folder by default. (You can specify another location within the **PatternTemplateFolderUrl** property of the DynamicPattern control if desired.)

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

PatternTemplates shipped with DES Dynamic Data

Here are the Pattern Templates installed by DES Dynamic Data:

File name	Description

DetailsView.ascx	Mimics the DetailsView control with a table that provides table rows for each data field specified. Its parts include optional headers and footers.
	`<tr>`
	`  <td>[Data Field Label]</td>`
	`  <td>[Data Field value]</td>`
	`</tr>`

DetailsViewButtons.ascx, DetailsViewLinkButtons.ascx	Used by the DetailsView.ascx file to provide buttons for the various modes (readonly, edit, and insert).
	`<tr>`
	`  <td>[Delete button]</td>`
	`  <td>[Edit button] [Insert button]</td>`
	`</tr>`

GridView.ascx	Mimics the GridView control with a table of rows and columns. Each row displays a data record. Each column displays a data field specified. It provides optional headers and footers. It also provides default output for special Templates within the ListView control such as GroupTemplate and EmptyDataTemplate.
	`<tr>`
	`  <td>[Field Title]</td>`
	`  <td>[Field Title]</td>`
	`</tr><tr>`
	`  <td>[Data Field Value]</td>`
	`  <td>[Data Field Value]</td>`
	`</tr>`

ListOfDetailViews.ascx	A list oriented Pattern Template where the data records are displayed like a DetailsView control.
ButtonsForList.ascx, LinkButtonsForList.ascx	Used by GridView.ascx, ListOfDetailViews.ascx, and any list oriented Pattern Template that you create to handle buttons for the various modes (readonly, edit, and insert).

ButtonsForSingle.ascx, LinkButtonsForSingle.ascx	Buttons for single-record oriented Pattern Templates that you create. Offers buttons in the various modes (readonly, edit, and insert) with HTML.
	`[Edit button] [Delete button] [Select button]`
	` [Details button]`

TwoColumnTableRow.ascx	Provides a table row with two columns much like a row in a DetailsView control. Often used in the ItemTemplate of a ListView control.
	`<tr>`
	`  <td>[Data Field Label]</td>`
	`  <td>[Data Field value]</td>`
	`</tr>`

TwoStacked.ascx	Displays two data fields and their labels in this format: `[label1] [data1] ` `[label2] [data2] `

Open each of these files for examples of using them with a DynamicPattern and to explore their Named Styles. If you choose to customize, you may prefer to use a duplicate of the file. You are encouraged to create your own as well. See "Customizing and creating your own PatternTemplates"

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

Style sheets

DES provides style sheet classes for these Pattern Templates within the **[web app]/DynamicData/Contents/PatternTemplateNamedStyles.css** file. This file is automatically loaded for you. Customize the styles as you need, or modify the Pattern Template file to use other style sheet classes. The file is heavily commented and each Pattern Template also provides documentation about its use of styles.

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

The DynamicPattern Control

The DynamicPattern control is used to add a Pattern Template file to the page, and to apply rules to it including:

- A list of data fields that are applied to DynamicControl and DynamicLabel controls contained within the Pattern Templates.
- A list of Named Styles, to adjust the formatting and appearance.
- Values for properties declared in the Pattern Template to further customize it.
- Select from alternative HTML output defined by the NamedParts feature of PartsPatternTemplates.

If you want to use a Pattern Template, you must add a DynamicPattern control to your form and specify the Pattern Template name in its **PatternTemplateName** property. Alternatively, you can use the DynamicListView and DynamicFormView controls. They collect all of the same information above, and internally create the DynamicPattern controls that interact with the Pattern Templates.

Click on any of these topics to jump to them:

- Using the DynamicPattern Control
- Adding a DynamicPattern Control
- Properties for the DynamicPattern Control
- Methods for the DynamicPattern Control

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

Using the DynamicPattern Control

A DynamicPattern control can be added to any Template within your DataBound control, such as `<ItemTemplate>` and `<HeaderTemplate>`. It generates some or all of the content for your DataBound control.

Click on any of these topics to jump to them:

- Properties that should always be considered
- Defining ItemInPattern objects in the ItemsInPattern collection
- Defining the Data Field names shown by the PatternTemplate
  - Only one data field
  - Creating the list of data fields from automatic scaffolding
  - Specifying a list of data fields
  - A NamedPart with more than one data field
- Buttons in PatternTemplates
- Passing text into PatternTemplates
- Using PatternTemplates to output content without data fields
- Adding HTML and other web controls between data fields
- Using Named Styles to adjust the formatting
- Passing properties to the Field Templates
- Passing properties to be used by the PatternTemplate's own code

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

Properties that should always be considered

Selecting a PatternTemplate

Always assign its **PatternTemplateName** property to the file name of the Pattern Template file you want. The ".ascx" file extension is optional. If your Pattern Template file is located in the **[web application]/ DynamicData/PatternTemplates** folder, it will be found. If it is another folder, specify the path to that folder within the **PatternTemplatesFolderUrl** property.

For the file "TwoStacked.ascx" within the **[web application]/DynamicData/PatternTemplates** folder:

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="TwoStacked" />
```

For the file "TwoStacked.ascx" within the **[web application]/PatternTemplates** folder:

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="TwoStacked" PatternTemplatesFolderUrl="~/PatternTemplates" />
```

Specifying the Data Entry Mode

The data entry mode must be identified within the **Mode** property when the controls are shown in Edit and Insert modes.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="TwoStacked" Mode="Edit" />
```

Working with a PartsPatternTemplate

The PartsPatternTemplate contains numerous possible ways to output HTML. It defines "Parts", which are the elements that make up a complex control, like HeaderRow, DataCell, and Buttons. You must specify which Part supplies the HTML output with the **PartsToGenerate** property.

The following are PartsPatternTemplates supplied with DES: **GridView.ascx**, **DetailsView.ascx**, and **ListOfDetailViews.ascx**.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="GridView" PartsToGenerate="HeaderRow" />
```

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="GridView" PartsToGenerate="DataRow" Mode="Edit" />
```

PartsToGenerate	Typical usage	ListView	FormView
ContainerOpeningTag	Start of the entire block of HTML.	LayoutTemplate or GroupTemplate	HeaderTemplate
ContainerClosingTag	End of the entire block of HTML	LayoutTemplate or GroupTemplate	FooterTemplate
DataRow	Output a single record	ItemTemplate, EditItemTemplate, InsertItemTemplate	ItemTemplate, EditItemTemplate, InsertItemTemplate
HeaderRow	Output titles or other headers	HeaderTemplate	HeaderTemplate
FooterRow	Output footer content	FooterTemplate	FooterTemplate
Buttons, ButtonsHeader, ButtonsFooter	Output buttons. This is normally used automatically when you use DataRow, HeaderRow, and FooterRow.	ItemTemplate, EditItemTemplate, InsertItemTemplate	ItemTemplate, EditItemTemplate, InsertItemTemplate

CONTINUED ON THE NEXT PAGE

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

PartsToGenerate	Typical usage	ListView	FormView
NoDataToShow	When no data is available	EmptyDataTemplate	EmptyDataTemplate
DetailView	Output all of the parts to make a single record interface.		ItemTemplate, EditItemTemplate, InsertItemTemplate
GroupContainer-OpeningTag and ClosingTag, GroupHeader, GroupFooter	ListView only. Support for the Group feature	LayoutTemplate when Grouping is enabled.	
GroupSeparator	Separator between items when using Groups	GroupSeparatorTemplate	
EmptyItem	ListView only. Support for the Group feature.	EmptyItemTemplate	

If you have defined "NamedParts" within the PartsPatternTemplate, you can specify default names to use within properties with this pattern: *PartType*NamedPart.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="GridView" PartsToGenerate="DataRow" Mode="Edit"
   ButtonsNamedPart="LinkButtons" />
```

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

Defining ItemInPattern objects in the ItemsInPattern collection

Most of the work is done by adding ItemInPattern objects to the **ItemsInPattern** property. The list of ItemInPattern objects determines the data fields to assign to DynamicControls.

ItemInPattern classes are objects subclassed from `PeterBlum.DES.DynamicData.BaseItemInPattern` that are used to describe the data fields and other elements output by a Pattern Template file. You add these objects to the **ItemsInPattern** property of DynamicPattern, DynamicFormView, and DynamicListView controls.

Here are a summary of the ItemsInPattern classes. Later topics go into detail.

- **DataFieldInPattern** class – Assigns a data field name to the DynamicControl, DynamicLabel, or DynamicColumnTitle control.
- **DataFieldListInPattern** class – Associates a list of DataFieldInPattern objects to a NamedPart that contains DynamicControls, DynamicLabels, and/or DynamicColumnTitles. NamedParts are associated with PartsPatternTemplates such as GridView.ascx, ListOfDetailsView.ascx, and DetailsView.ascx.
- **ScaffoldInPattern** class – Create a list of DataFieldInPattern objects using Automatic Scaffolding. Each uses a separate ItemTemplate instance for its data field name. Often you can avoid using these by specifying a value in the **AutoScaffoldDataFields** property of DynamicPattern, DynamicFormView, or DynamicListView controls.
- **ButtonsInPattern** class – Adds buttons through a Buttons NamedPart on a PartsPatternTemplate. Often you can avoid using this by specifying a value in the **AutoGenerateButtons** property of DynamicPattern, DynamicFormView, or DynamicListView controls.
- **LabelInPattern** class – Assigns text to a DynamicLabel control. Allows the DynamicLabel to be a placeholder for any text. (DataFieldInPattern also assigns text, but it specifically assigns a data field name.)
- **NoDataFieldsInPattern** class – Uses a NamedPart to provide additional formatting that lacks DynamicControls and DynamicLabels. It is effectively a User Control that supports the NamedStyles of the DynamicPattern control.
- **TemplateNotInPattern** class – Declares HTML and web controls, including DynamicControls and DynamicLabels, without having to modify the Pattern Template.

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

Defining the Data Field names shown by the PatternTemplate

The purpose of a Pattern Template is to describe the HTML for a page without defining the data fields for each DynamicControl. The DynamicPattern is responsible for attach the data fields at runtime. There are several ways you can provide these fields.

Only one data field

If you have only one data field to declare, use the **DataField** property directly on the DynamicPattern control.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="MyPatternTemplate" DataField="columnname" />
```

This applies mostly to Pattern Templates subclassing from `PeterBlum.DES.DynamicData.DataFieldsPatternTemplate` that have a single DynamicControl. If they also have a DynamicLabel, use the **AssociatedControlID** property on the DynamicLabel to connect it to the DynamicControl.

Creating the list of data fields from automatic scaffolding

If you want the same list of data fields as determined by automatic scaffolding, simply declare the `PeterBlum.DES.DynamicData.ScaffoldInPattern` object in the **ItemsInPattern** property of the DynamicControl.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="MyPatternTemplate" >
   <ItemsInPattern>
      <desDD:ScaffoldInPattern />
   </ItemsInPattern>
</desDD:DynamicPattern>
```

Alternatively, you can keep the **ItemsInPattern** property empty and the DynamicPattern will automatically scaffold because its **AutoScaffoldDataField** defaults to `Auto`.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="MyPatternTemplate" >
</desDD:DynamicPattern>
```

This technique has limitations. Because you cannot declare individual DataFieldInPattern objects, you cannot set any properties associated with DataFieldInPattern objects, including **ControlIDInPattern**, **PositionInPattern**, **NamedPart**, or any properties to pass along to the Field Templates they generate (via these properties: **UIHint**, **DefaultPropertyList**, **ReadOnlyUIHint**, **ReadOnlyPropertyList**, **EditUIHint**, **EditPropertyList**, **InsertUIHint**, and **InsertPropertyList**.)

Specifying a list of data fields

If you have a list of data fields to declare, add `PeterBlum.DES.DynamicData.DataFieldInPattern` objects to the **ItemsInPattern** property of the DynamicPattern control. Define the data field in the **DataField** property of the DataFieldInPattern object. The list defines the order they are applied against the DynamicControls within the Pattern Template.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="MyPatternTemplate" >
   <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="columnname1" />
      <desDD:DataFieldInPattern DataField="columnname2" />
      <desDD:DataFieldInPattern DataField="columnname3" />
   </ItemsInPattern>
</desDD:DynamicPattern>
```

Note: By adding at least one item to ItemsInPattern, the DynamicPattern control will no longer populate it for you using Automatic Scaffolding.

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

There are properties on DataFieldInPattern to help you express the exact DynamicControl or DynamicLabel that receives the data fields: **ControlIDInPattern** to specify by the ID property on the control or **PositionInPattern** to specify it by its order.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="MyPatternTemplate" >
   <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="columnname1"
           ControlIDInPattern="ControlID1" />
      <desDD:DataFieldInPattern DataField="columnname2"
           ControlIDInPattern="ControlID2" />
      <desDD:DataFieldInPattern DataField="columnname3"
           ControlIDInPattern="ControlID3" />
   </ItemsInPattern>
</desDD:DynamicPattern>
```

When using a Pattern Template that subclasses from `PeterBlum.DES.DynamicData.PartsPatternTemplate`, be aware that it will choose from various Templates based on the mode in it DataRow and DataCell parts. Readonly uses `<Template>`; Edit mode uses `<EditTemplate>`; and Insert mode uses `<InsertTemplate>`. If they are not declared, they fallback to `<EditTemplate>` or `<Template>`. So make sure your ControlID reflects the correct Template.

Additionally with PartsPatternTemplates, you can specify a NamedPart by setting the name of the NamedPart in the **NamedPart** property on the DataFieldInPattern object.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="GridView" >
   <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="columnname1"
           NamedPart="Format2" />
      <desDD:DataFieldInPattern DataField="columnname2" />
   </ItemsInPattern>
</desDD:DynamicPattern>
```

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

A NamedPart with more than one data field

NamedParts are declared on PartsPatternTemplates to allow alternative content. Many times, they offer multiple DynamicControls, each needing its own data field name to be assigned.

Here is the `<DataCell>` part from GridView. It has two NamedParts.

```
<DataCell NamedStyle="DataCell" CssClass="<%#cCssDataCell %>"
      Style="vertical-align:top;" >
   <NamedParts>
      <desDD:DataCellNamedPatternPart Name="TwoStacked">
         <Template>
            <desDD:DynamicControl ID="DynamicControl1" runat="server" /><br />
            <desDD:DynamicControl ID="DynamicControl2" runat="server" />
         </Template>
      </desDD:DataCellNamedPatternPart>
      <desDD:DataCellNamedPatternPart Name="ShowLabel">
         <Template>
            <desDD:DynamicLabel ID="DynamicLabel1" runat="server" />
         </Template>
      </desDD:DataCellNamedPatternPart>
   </NamedParts>
</DataCell>
```

Use the `PeterBlum.DES.DynamicData.DataFieldListInPattern` object to specify the NamedPart within its **NamedPart** property. Then add a list of DataFieldInPattern objects (from the previous topic) to the **ItemsInPattern** collection of DataFieldListInPattern.

Here is the DynamicPattern using "TwoStacked" from the above example.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="GridView" PartsToGenerate="DataCell" >
   <ItemsInPattern>
      <desDD:DataFieldListInPattern NamedPart="TwoStacked" />
         <ItemsInPattern>
            <desDD:DataFieldInPattern DataField="columnname1" />
            <desDD:DataFieldInPattern DataField="columnname2" />
         </ItemsInPattern>
      </desDD:DataFieldListInPattern>
   </ItemsInPattern>
</desDD:DynamicPattern>
```

The child DataFieldInPattern objects still support the **ControlIDInPattern** and **PositionInPattern** properties to assist in matching the data fields to their DynamicControls.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="GridView" PartsToGenerate="DataCell" >
   <ItemsInPattern>
      <desDD:DataFieldListInPattern NamedPart="TwoStacked" />
         <ItemsInPattern>
            <desDD:DataFieldInPattern DataField="columnname1"
               ControlIDInPattern="DynamicControl1" />
            <desDD:DataFieldInPattern DataField="columnname2"
               ControlIDInPattern="DynamicControl2" />
         </ItemsInPattern>
      </desDD:DataFieldListInPattern>
   </ItemsInPattern>
</desDD:DynamicPattern>
```

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

Buttons in PatternTemplates

Pattern Templates can be created for other content than showing data fields and their labels. They may show buttons, text, and more.

When working with DataBound controls, buttons are shown based on the mode of the control. Read only generally shows commands to start editing, like New and Edit, while Edit and Insert modes show the Save and Cancel buttons. The `PeterBlum.DES.DynamicData.DynamicButtonsPatternTemplate` class describes the user interface for buttons that create, edit, delete, save and cancel a record. DynamicButtonsPatternTemplate uses information about the DataBound control's mode to display the correct buttons. It also uses information coming from the DataSource and DataBound Control Adapter to determine if certain buttons are available.

DES Dynamic Data provides several Pattern Templates containing buttons: ButtonsForList.ascx, LinkButtonsForList.ascx, ButtonsForSingle.ascx, LinkButtonsForSingle.ascx, DetailsViewButtons.ascx, and DetailsViewLinkButtons.ascx.

Use the DynamicPattern control to load these, only you don't specify any data fields.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
  PatternTemplateName="DetailsViewLinkButtons" />
```

When using a PartsPatternTemplate, you may have declared NamedParts that use other Pattern Templates by specifying the name within the **PatternTemplateName** property. Here is code taken from **DetailsView.ascx** that has two NamedParts called "Buttons" and "LinkButtons" that declare DynamicPattern controls for "DetailsViewButtons" and "DetailsViewLinkButtons".

```
<Buttons Tag="None" PatternTemplateName="DetailsViewLinkButtons" >
   <NamedParts>
      <desDD:ButtonsNamedPatternPart Name="LinkButtons"
         PatternTemplateName="DetailsViewLinkButtons" />
      <desDD:ButtonsNamedPatternPart Name="Buttons"
         PatternTemplateName="DetailsViewButtons" />
   </NamedParts>
</Buttons>
```

Use the `PeterBlum.DES.DynamicData.ButtonsInPattern` object to apply one of these NamedParts by specifying the name of the NamedPart (not the file name of the Pattern Template) in its **NamedPart** property.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="GridView" PartsToGenerate="DataRow" >
   <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="columnname1" />
      <desDD:DataFieldInPattern DataField="columnname2" />
      <desDD:DataFieldInPattern DataField="columnname3" />
      <desDD:ButtonsInPattern NamedPart="Button" />
   </ItemsInPattern>
</desDD:DynamicPattern>
```

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

Passing text into PatternTemplates

A Pattern Template must be flexible so that you don't have to re-engineer its patterns with each situation. One common case is that of text it shows. For example, a title changes for each situation. You should be able to pass the desired text for the title to the Pattern Template and it will appear in the right spot in the pattern.

To do this, add a DynamicLabel control within your Pattern Template and give it a unique ID value. Use the LabelInPattern object to associate text with that label like this:

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="GridView" PartsToGenerate="HeaderRow" >
   <ItemsInPattern>
      <desDD:LabelInPattern Text="Welcome" ControlIDInPattern="TitleLabel" />
   </ItemsInPattern>
</desDD:DynamicPattern>
```

An alternative technique is to establish a property on the Pattern Template that will be assigned the title. Let's suppose you define the property **Title** in your Pattern Template. Your Template's ASP.NET markup can output the property like this:

```
<% =Title %>
```

or

```
<asp:Label id="TitleLabel" runat="server" Text="<%# Title %>" />
```

Within the DynamicPattern control, specify the same property name and its value.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="MyPatternTemplate" Title="Welcome" >
```

For an example of this setup, see the "TwoDataFields" NamedPart in `<DataCell>` and `<HeaderRow>` Part in the **DetailsView.ascx** Pattern Template field.

For more on setting properties, see "Passing properties to be used by the PatternTemplate's own code".

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

Using PatternTemplates to output content without data fields

A NamedPart of a PartsPatternTemplate can hold any HTML and web controls. When it does not contain any DynamicControls, DynamicLabels, or DynamicColumnTitle controls, use the NoDataFieldsInPattern object to specify its part.

In this example, an image is defined in the `<DataCell>` NamedParts called GraphicIntro.

```
<DataCell properties >
  <NamedParts>
    <desDD:DataCellNamedPatternPart Name="GraphicIntro">
    <Template>
      <asp:Image ID="Image1" runat="server" ImageUrl="~/Images/Logo.jpg" />
    </Template>
    </desDD:DataCellNamedPatternPart>
  </NamedParts>
</DataCell>
```

It is used as the first element in a data row, followed by the actual data fields.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
  PatternTemplateName="GridView" PartsToGenerate="DataRow" >
  <ItemsInPattern>
    <desDD:NoDataFieldsInPattern NamedPart="GraphicIntro" />
    <desDD:DataFieldInPattern DataField="columnname1" />
    <desDD:DataFieldInPattern DataField="columnname2" />
    <desDD:DataFieldInPattern DataField="columnname3" />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

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

Adding HTML and other web controls between data fields

Sometimes you are creating a list of data fields to be generated by a Pattern Template and identify one that just needs its own formatting, not declared (and shouldn't be added to) the Pattern Template. You can use the `PeterBlum.DES.DynamicData.TemplateNotInPattern` object to define the desired HTML and web controls, including DynamicControls. In this case, you explicitly define the **DataField** property on any DynamicControls within the template.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="MyPatternTemplate" >
   <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="columnname1" />
      <desDD:DataFieldInPattern DataField="columnname2" />
      <desDD:DataFieldInPattern DataField="columnname3" />
      <desDD:TemplateNotInPattern>
         <Template>
            <tr>
               <td align='right'>
                  <desDD:DynamicControl id="StartRange" runat="server"
                     DataField="columnname4" UIHint="Currency" /> - 
                  <desDD:DynamicControl id="EndRange" runat="server"
                     DataField="columnname5" UIHint="Currency" />
               </td>
            </tr>
         </Template>
      </desDD:TemplateNotInPattern>
   </ItemsInPattern>
</desDD:DynamicPattern>
```

You can also use the TemplateNotInPattern object to add HTML between the HTML output by your Pattern Template by declaring it either as a **Template** (which allows ASP.NET markup) or **HTML** property (which allows a string of HTML.)

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="MyPatternTemplate" >
   <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="columnname1" />
      <desDD:TemplateNotInPattern HTML=" " />
      <desDD:DataFieldInPattern DataField="columnname2" />
      <desDD:TemplateNotInPattern HTML=" " />
      <desDD:DataFieldInPattern DataField="columnname3" />
   </ItemsInPattern>
</desDD:DynamicPattern>
```

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

Using Named Styles to adjust the formatting

Evaluate the Pattern Template file to see if you can use Named Styles.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
     PatternTemplateName="MyPatternTemplate" >
  <NamedStyles>
     <desDD:NamedStyle Name="LabelControl" CssClass="SmallLabel" />
     <desDD:NamedStyle Name="DataColumn" Style="width:200px;white-space:nowrap;" />
     <desDD:NamedStyle Name="DataColumn" DataFields="QuantityPerUnit"
          Style="white-space:nowrap;" />
  </NamedStyles>
</desDD:DynamicPattern>
```

Because a NamedStyle can evaluate data field, category, and user role names, you may specify several NamedStyles with the same name, but different combinations of the other evaluators. It will always use the most specific match over a less specific match. For example, in the above example, it uses the third NamedStyle when "DataColumn" is being generated. If that is not generating output for "QuantityPerUnit", it then uses the second NamedStyle. If two NamedStyles could match and have the same number of evaluators, it uses the order specified in the **NamedStyles** property.

See the following topics:

- Named Styles: Your pages control the formatting
- NamedStyles on the DetailsView PatternTemplate
- NamedStyles on the GridView PatternTemplate
- NamedStyles on the ListOfDetailViews PatternTemplate
- NamedStyles on the TwoColumnTableRow PatternTemplate
- NamedStyles on the TwoStacked PatternTemplate
- NamedStyles on these PatternTemplate
- NamedStyles on the ButtonsForList and LinkButtonsForList PatternTemplate
- NamedStyles on the ButtonsForSingle and LinkButtonsForSingle PatternTemplate
- The DynamicStylesTag Control
- The NamedStyle Class

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

Passing properties to the Field Templates

As you know, the DynamicControl uses a Field Template to generate its HTML output. It has properties to specify which template (**UIHint**) and to set other properties found on the Field Template (using "Attributes").

For example, here are a few of the properties offered on the Integer_Edit.ascx Field Template: SmartColumns and ValueWhenBlank. You would normally assign them to the DynamicControl as if they are its own properties, like this:

```
<desDD:DynamicControl id="DC1" runat="server"
   SmartColumns="Always" ValueWhenBlank="Fill in" />
```

Note: The Customizer control also sets many properties of Field Templates.

These features are available when using Pattern Templates. However, it doesn't make sense to declare them on the DynamicControls found within the Pattern Template file because they don't know which data field they are setting up.

Instead, these values should be declared wherever you specify the **DataField** property. When using Pattern Templates, these properties are set in the `PeterBlum.DES.DynamicData.DataFieldInPattern` object. You have seen DataFieldInPattern objects throughout the previous sections. Here are properties on a DataFieldInPattern object that setup **UIHint** and other properties on the Field Template.

- **UIHint**, **ReadOnlyUIHint**, **EditUIHint**, and **InsertUIHint** – UIHint sets the same named-property on the Field Template to override which Field Template file is selected by default. When using DynamicPattern controls, you don't always know what the mode will be (readonly, edit, and insert). Sometimes the mode requires a different UIHint value. As a result, you can specify mode-specify UIHints with the **ReadOnlyUIHint**, **EditUIHint**, and **InsertUIHint** properties.
- **DefaultPropertyList**, **ReadOnlyPropertyList**, **EditPropertyList**, **InsertPropertyList** – Declare any properties that you need to pass along to the Field Template. These properties are actually declared in the Field Template class itself. For example, any textual value shown in readonly mode offers the **HtmlEncode** property to determine if HTML tags are shown as HTML or as raw text. Like with UIHints, there are mode-specific properties available.

  The format of these properties is a string containing ASP.NET markup, just as you would enter it into the control. Make sure you enclose each property in single quotes. Here's the overall pattern:

  ```
  DefaultPropertyList="PropertyName1='propertyvalue' PropertyName2='propertyvalue'"
  ```

  Hint: If you need single quotes within your property values, enter them as "&#39".

Example

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
  PatternTemplateName="MyPatternTemplate" >
  <ItemsInPattern>
    <desDD:DataFieldInPattern DataField="Description" UIHint="FancyText"
      ReadOnlyPropertyList="HtmlEncode='true'" />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

This example shows a single DataFieldInPattern object. Since you can also setup a single data field using the **DataField** property on DynamicPattern, do not use the **DataField** property if you intend to set any of the **UIHint** or **PropertyList** properties. *The DataField property is a shortcut, covering the most basic setup cases.*

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

Passing properties to be used by the PatternTemplate's own code

Pattern Template can expose public properties that let you customize their behavior. For example, the **DetailsView.ascx** Pattern Template has the Boolean property **HeaderLabelColumnHTML** which specifies the text of the label in the HeaderRow above the Label column.

[C#]

```
public string HeaderLabelColumnHTML
{
   get { return Host.GetAttributeAsString("HeaderLabelColumnHTML",
      cDefaultHeaderLabelColumnHTML); }
}
private const string cDefaultHeaderLabelColumnHTML = "Name";
```

[VB]

```
Public ReadOnly Property HeaderLabelColumnHTML() As String
   Get
      Return Host.GetAttributeAsString("HeaderLabelColumnHTML",
         cDefaultHeaderLabelColumnHTML)
   End Get
End Property
Private Const cDefaultHeaderLabelColumnHTML As String = "Name"
```

The value of this property is assigned on the DynamicPattern control and retrieved by your property like this:

```
<desDD:DynamicPattern id="DP1" runat="server"
   PatternTemplateName="DetailsView" HeaderLabelColumnHTML="Field Name" />
```

Just declare the property name and value within the ASP.NET markup of the DynamicPattern control. Some of the properties it already knows about and lists in the Properties Editor of Visual Studio. Others – especially those you created – are not known but still can be specified.

Passing Properties specific to an ItemInPattern object

When using a PartsPatternTemplate, these properties can be passed along on individual DataFieldInPattern, DataFieldListInPattern, ButtonsInPattern, NoDataFieldsInPattern, and TemplateNotInPattern objects to be used by code within the Pattern Template in their **OnPartCreating** or **OnPartCreated** event handler methods.

The Pattern Template must call the `GetAttributeAsNative()` or `GetAttributeAsString()` methods on the *Args* parameter object passed to the **OnPartCreating** or **OnPartCreated** event handlers. Good documentation on the property names it supports is vital.

When the page developer identifies such a property, they specify the property name and value within the ASP.NET markup of the *xyz*InPattern object. In this example, the "ColumnTitle" NamedPart supports a property named "Title" that takes a string.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="MyPatternTemplate" >
   <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="columnname1" UIHint="FancyText"
         NamedPart="ColumnTitle" Title="my title" />
   </ItemsInPattern>
</desDD:DynamicPattern>
```

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

Adding a DynamicPattern Control

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

1. Review the existing Pattern Templates to determine the right one. You may need to create one if none are a match. See:

   - Using the PatternTemplates shipped with DES Dynamic Data
   - Customizing and creating your own PatternTemplates

   By default, Pattern Templates go in the **[web application]/DynamicData/PatternTemplates** folder.

   Suggestion: Keep the PatterrnTemplate file open in an editor as you work.

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

3. Add an DynamicPattern control to the page. It must go in a Template area of a DataBound control, such as ItemTemplate on ListView, a TemplateField object on GridView, or within another Pattern Template file.

   Visual Studio and Visual Web Developer Design Mode Users

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

   Text Entry Users

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

```
<desDD:DynamicPattern id="[YourControlID]" runat="server" />
```

   Programmatically creating the DynamicPattern control

   - Identify the control which you will add the DynamicPattern control to its **Controls** collection. Like all ASP.NET controls, the DynamicPattern control can be added to any control that supports child controls, including Templates.
   - Create an instance of the DynamicPattern control class. The constructor takes no parameters.
   - Assign the **ID** property.
   - Add the DynamicPattern control to the **Controls** collection.

   In this example, the DynamicPattern is created with an **ID** of "DynamicPattern1". It is added to PlaceHolder1.

   [C#]

```
PeterBlum.DES.DynamicData.DynamicPattern vDynamicPattern =
   new PeterBlum.DES.DynamicData.DynamicPattern();
vDynamicPattern.ID = "DynamicPattern1";
container.Controls.Add(vDynamicPattern);
```

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

   [VB]

```
Dim vDynamicPattern As PeterBlum.DES.DynamicData.DynamicPattern = _
   New PeterBlum.DES.DynamicData.DynamicPattern()
vDynamicPattern.ID = "DynamicPattern1"
container.Controls.Add(vDynamicPattern)
```

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

CONTINUED ON THE NEXT PAGE

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

4. Set the **PatternTemplateName** property to the file name of the Pattern Template file. The file extension (".ascx") is optional.

   If your Pattern Template file is not in the **[web application]/DynamicData/PatternTemplates** folder, set the URL to the correct folder within the **PatternTemplatesFolderUrl** property.

Guidelines for setting properties

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

5. Determine the data entry mode: ReadOnly, Edit, or Insert. If it is Edit or Insert, assign that value to the **Mode** property.
6. When the Pattern Template file inherits from `PeterBlum.DES.DynamicData.PartsPatternTemplate`, set the **PartsToGenerate** property to the desired part. See "Working with a PartsPatternTemplate".
7. If this DynamicPattern control has an explicit usage, such as header, footer, or button, set the usage in the **PatternTemplateUsage** property. While not required, if you are using TemplateNotInPattern, this lets it select the correct Template. It also provides error checking in case you select a Pattern Template file that has a different usage.
8. Add the names of Data Fields in several ways:

   - Add `PeterBlum.DES.DynamicData.DataFieldInPattern` objects to the **ItemsInPattern** collection.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
    PatternTemplateName="MyPatternTemplate" >
    <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="columnname1" />
      <desDD:DataFieldInPattern DataField="columnname2" />
      <desDD:DataFieldInPattern DataField="columnname3" />
    </ItemsInPattern>
</desDD:DynamicPattern>
```

   - Use automatic scaffolding by leaving the **ItemsInPattern** collection empty or by setting **AutoScaffoldDataFields** to `True`.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
    PatternTemplateName="MyPatternTemplate" AutoScaffoldDataFields="True" >
</desDD:DynamicPattern>
```

   - If you only have one Data Field, you can set its name directly on the **DataField** property of DynamicPattern, as long as you don't need to set any other properties available to you on the `PeterBlum.DES.DynamicData.DataFieldInPattern` class.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
    PatternTemplateName="MyPatternTemplate" DataField="Fieldname" >
</desDD:DynamicPattern>
```

   Note: If the Pattern Template is based on PeterBlum.DES.DynamicData.NoDataFieldsPatternTemplate or PeterBlum.DES.DynamicData.DynamicButtonsPatternTemplate, you can skip this step.

   See "Defining the Data Field names shown by the PatternTemplate".

CONTINUED ON THE NEXT PAGE

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

9. You can add other objects to the **ItemsInPattern** collection to change the default formatting. See:

   - Buttons in PatternTemplates
   - Passing text into PatternTemplates
   - Using PatternTemplates to output content without data fields
   - Adding HTML and other web controls between data fields

10. Evaluate the formatting of the Pattern Template. If it needs adjustment to style sheets or specific style attributes, see if that element supports a Named Style. It will use a `<desDD:DynamicStylesTag>` control or a call to a method that supports Named Styles.

11. Add `PeterBlum.DES.DynamicData.NamedStyle` objects to the **NamedStyles** collection for each Named Style that you want to override.

    See "Named Styles: Your pages control the formatting".

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
  PatternTemplateName="TwoColumns" DataField="ProductName" >
  <NamedStyles>
    <desDD:NamedStyle Name="LabelColumn" Style="width:150px;" />
    <desDD:NamedStyle Name="Row" CssClass="CustomRowStyle" />
  </NamedStyles>
</desDD:DynamicPattern>
```

12. Review the public properties exposed with the Pattern Template file's code.

    If there are any property values you want to change, add their property name and value to the DynamicPattern control's markup or through its `SetAttributeAsNative()` method. See "Passing properties to be used by the PatternTemplate's own code".

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

Properties for the DynamicPattern Control

`PeterBlum.DES.DynamicData.DynamicPattern` control is subclassed from `System.Web.UI.Control`.

Click on any of these topics to jump to them:

- Template Lookup Properties
- Data Properties
- Appearance Properties
- PartsPatternTemplate Properties
- Default NamedParts Properties
- Scaffold Caching Properties
- Behavior Properties

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

Template Lookup Properties

These properties appear in the "Template Lookup" category of the Properties Editor.

- **PatternTemplateName** (string) – This required property determines which Pattern Template file to use. The file extension ".ascx" is optional. This property should not have a URL. If the Pattern Template file is not in the **[web application]/DynamicData/PatternTemplates** folder, use the **PatternTemplatesFolderUrl** property to specify the URL.
- **PatternTemplatesFolderUrl** (string) – The URL to the folder containing the Pattern Template file. If the file is in the **[web application]/DynamicData/PatternTemplates** folder, this property can be unassigned.

  It defaults to "".
- **Mode** (enum System.Web.UI.WebControls.DataBoundControlMode) – Determines the data entry mode for the Pattern Template: ReadOnly, Edit, or Insert. It defaults to DataBoundControlMode.ReadOnly.
- **PatternTemplateUsage** (enum PeterBlum.DES.DynamicData.PatternTemplateUsage) – Imposes a limit on the types of Pattern Templates that can be used. At design time, it gives the Properties Editor a specific list of Pattern Templates in the **PatternTemplateName** property. At runtime, it causes an exception if the Pattern Template requested does not match this value.

  If you are using a TemplateNotInPattern object in the **ItemsInPattern** collection, it offers three templates: Template, HeaderTemplate, and FooterTemplate. The **PatternTemplateUsage** property tells TemplateNotInPattern which template to use.

  The enumerated type PatternTemplateUsage has these values:

  - `Any` - Any Pattern Template is allowed. This is the default.
  - `Parts` - Used by PartsPatternTemplates and supported by DynamicListView/DynamicFormView. If you select `Parts` and the Pattern Template's **Usage** is either `PartsForSingle` or `PartsForList`, the Pattern Template will match.
  - `PartsForList` - Used by PartsPatternTemplates and supported by DynamicListView. Describes a list style interface.
  - `PartsForSingle` - Used by PartsPatternTemplates and supported by DynamicFormView. Describes a single record style interface
  - `Buttons` - Used by DynamicButtonsPatternTemplates.
  - `Data` - Contains DynamicControls, DynamicLabels, or DynamicColumnTitles. Can be used in DataFieldsPatternTemplates
  - `Header` - For Header. Can contain data fields.
  - `Footer` - For footer. Can contain data fields.
  - `NoDataFields` - Used by NoDataFieldsPatternTemplates. There should not be any DynamicControls or DynamicLabels that need their DataField property assigned.

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

Data Properties

These properties appear in the "Data" category of the Properties Editor.

- **ItemsInPattern** (PeterBlum.DES.DynamicData.ItemsInPattern) – Use this collection to identify the data fields that will be assigned to DynamicControls found in the Pattern Template. It can also specify formatting available through NamedParts of PartsPatternTemplates and TemplateNotInPattern objects.

  See "Using the DynamicPattern Control".

  If you need to programmatically find a specific object within ItemsInPattern, there are two methods available on the DynamicPattern control:

  - For DataFieldInPattern objects, use `FindByDataField("datafieldname")`. See "GetAttributeAsInteger() method".
  - For all objects, assign a unique value in the ItemInPattern object's ID property when the object is first created or in the ASP.NET markup. Then use `FindByID("id")` and typecast to the expected class. See "FindByID() method".

- **DataField** (string) – A data field name to assign to a DynamicControl within the Pattern Template. Use instead of the **ItemsInPattern** property when you have one data field name to define and don't need any other feature of the DataFieldInPattern object.

  This property simplifies setup when you only have a single column name and no other attributes to pass to the Pattern Template.

  If specified and the **ItemsInPattern** property is also used, this value will become the first element of ItemsInPattern as a DataFieldInPattern object.

  It defaults to "".

- **AutoScaffoldDataFields** (enum PeterBlum.DES.DynamicData.AutoScaffoldDataFields) – Determines if the **ItemsInPattern** collection is automatically populated with Data Fields using the Automatic Scaffolding rules.

  When in use, this adds DataFieldInPattern objects to the ItemsInPattern collection by using the Scaffolding rules. When ItemsInPattern already has objects, scaffolded objects are added at the end of the collection.

  This property is ignored when the **PartsToGenerate** property does not reflect a data oriented part, such as `ContainerOpeningTag` and `Buttons`. It is also ignored when the **DataField** property is assigned.

  The enumerated type AutoScaffoldDataFields has these values:

  - `True` – Use automatic scaffolding to populate **ItemsInPattern**.
  - `False` – Do not modify the **ItemsInPattern** collection.
  - `WhenEmpty` - The **ItemsInPattern** collection must be empty to use automatic scaffolding. This is the default.

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

Appearance Properties

- **NamedStyles** (PeterBlum.DES.DynamicData.NamedStyles) – A list of NamedStyle objects. These are passed to the Pattern Template.

  Named Styles allow the user to deliver formatting rules into your Pattern Template. You declare various HTML tags that they can format with a DynamicStylesTag control, or by calling the `GetTagAttributes()` method. These take a name which is used to lookup a NamedStyle object from this list. (You don't have to search this list yourself.)

  For more details, see "Named Styles: Your pages control the formatting".

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

PartsPatternTemplate Properties

These properties appear in the "PartsPatternTemplate" category of the Properties Editor.

- **PartsToGenerate** (enum PeterBlum.DES.DynamicData.PatternPartsToGenerate) – Determines which part to output. There are a number of parttypes that don't match the Part properties within the DynamicPartsPatterns control. They alter the construction further.

  Often several DynamicPattern controls will be defined against the same Pattern Template to output the parts in the desired order. TemplateNotInPattern objects also use this to select between the three sources of HTML: standard, Footer, and footer. It selects Footer only when FooterRow, FooterCell,

  or FooterButtons is set. It selects Footer only when FooterRow, FooterCell, or FooterButtons is set. Otherwise it selects the standard HTML.

  The enumerated type PatternPartsToGenerate has these values.

  - `ContainerOpeningTag` - Generate the Container's opening tag
  - `DataRow` - Uses Mode to determine Template, EditTemplate, or Insertttemplate
  - `DataCell` - Uses Mode to determine Template, EditTemplate, or Insertttemplate
  - `Buttons` - Buttons within a Data Row
  - `HeaderRow` - Header row
  - `HeaderCell` - A single field within a HeaderRow
  - `HeaderButtons` - Header for Buttons within a Header Row. Used when AutoGenerateButtons is setup
  - `FooterRow` - Footer row
  - `FooterCell` - A single field within a FooterRow
  - `FooterButtons` - Footer for Buttons within a Footer Row. Used when AutoGenerateButtons is setup
  - `ContainerClosingTag` - Generate the Container's closing tag
  - `NoDataToShow` - When there are no rows of data to show. Generally this is a message to that face.
  - `DataRowSeparator` - For the ItemSeparatorTemplate of the ListView, this is a row that separates each DataRow. It can contain the entire HTML or let DataRowSeparatorCells define the appearance for each cell associated with a DataFieldInPattern object.
  - `DataRowSeparatorCell` - For the ItemSeparatorTemplate of the ListView, this a cell within a row that separates each DataRow. It only appears when DataRowSeparator indicates its use. One is generated for each DataFieldInPattern object.
  - `DataRowReadOnly` - Explicitly request a Data row by readonly mode. Will also explicity use the readonly version of the DataCell.
  - `DataRowEdit` - Explicitly request a Data row by edit mode. Will also explicity use the readonly version of the DataCell. Falls back to readonly mode if not declared.
  - `DataRowInsert` - Explicitly request a Data row by insert mode. Will also explicity use the readonly version of the DataCell. Falls back to edit then readonly mode if not declared.
  - `DataCellReadOnly` - Explicitly request a Data Cell by readonly mode.
  - `DataCellEdit` - Explicitly request a Data field by edit mode. Falls back to readonly mode if not declared
  - `DataCellInsert` - Explicitly request a Data field by insert mode. Falls back to edit then readonly mode if not declared
  - `DetailView` - Generate a full record. This is generally used by a Detail view. It includes container opening tag, header row, data row, footer row, container closing tag.

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

- `DetailViewNoHeader` - Generate a full record without the header. This is generally used by a Detail view. It includes container opening tag, data row, footer row, container closing tag.
- `DetailViewNoFooter` - Generate a full record without the footer. This is generally used by a Detail view. It includes container opening tag, header row, data row, container closing tag.
- `DetailViewNoHeaderFooter` - Generate a full record without the header or footer. This is generally used by a Detail view. It includes container opening tag, data row, container closing tag.
- `GroupContainerOpeningTag` - For ListView's Grouping feature (automatic GroupTemplate generation). Defines the opening HTML tag that encloses a group.
- `GroupContainerClosingTag` - For ListView's Grouping feature (automatic GroupTemplate generation). Defines the closing HTML tag that encloses a group.
- `GroupHeader` - For ListView's Grouping feature (automatic GroupTemplate generation). HTML that separates the Group's opening tag from GroupContainer from the first Item of actual data. It is often used for titles. It does not support DynamicControls.
- `GroupFooter` - For ListView's Grouping feature (automatic GroupTemplate generation). HTML that separates the Group's closing tag from GroupContainer from the last Item of actual data. It does not support DynamicControls.
- `GroupSeparator` - For ListView's Grouping feature (automatic GroupTemplate generation). HTML that separates groups from each other. Associated with the ListView GroupSeparatorTemplate feature.
- `EmptyItem` - For ListView's Grouping feature. It sets up the value of the EmptyItemTemplate property. EmptyItemTemplate defines the appearance where an item would normally go in a group, but there is no data for that item. It can contain the entire HTML or let EmptyItemCells define the appearance for each cell associated with a DataFieldInPattern object.
- `EmptyItemCell` - For the EmptyItemTemplate of the ListView, this a cell within a row that describes an record without any data. It only appears when EmptyItem indicates its use. One is generated for each DataFieldInPattern object.
- `Unknown` - The part is not associated with the DynamicPartsPatterns control, such as from a TemplateNotInPattern object.

It defaults to `PatternPartsToGenerate.DataRow`.

- **DataRowAppearance** (enum PeterBlum.DES.DynamicData.DataRowAppearance) – Determines the style sheet class and NamedStyle rule apply to the container tag around the DataRow and DataCell Parts.

  The enumerated type DataRowAppearance has these values:

  - `Standard` – Use the CssClass and NamedStyle properties on DataRow and DataCell Parts. This is the default.
  - `Selected` – Use the SelectedCssClass and SelectedNamedStyle properties on DataRow and DataCell Parts.
  - `Alternative` – Use the AlternativeCssClass and AlternativeNamedStyle properties on DataRow and DataCell Parts.

- **AutoGenerateButtons** (enum PeterBlum.DES.DynamicData.AutoGenerateButtons) – Determines where buttons are shown when generating a DataRow Part, or where a header or footer for buttons is positioned when generating a HeaderRow or FooterRow Part.

  The enumerated type values are:

  - `First` – Show buttons before the DataCells.
  - `Last` – Show the buttons after the DataCells. This is the default.
  - `FirstLast` - Before and after the DataCells.
  - `None` - Do not auto generate. Often the user declares a ButtonsInPattern object to generate buttons when set to `None`.

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

Default NamedParts Properties

These properties appear in the "Default NamedParts" category of the Properties Editor.

Each Part in a PartsPatternTemplate can use a NamedPart instead of the part declared on the DynamicPartsPatterns control. The following chart indicates the properties that define a NamedPart by its name.

Part Type	Property name
Container	ContainerNamedPart
HeaderRow	HeaderRowNamedPart
HeaderCell	HeaderCellNamedPart
HeaderButtons	HeaderButtonsNamedPart
DataRow	DataRowNamedPart
DataCell	DataCellNamedPart
Buttons	ButtonsNamedPart
FooterRow	FooterRowNamedPart
FooterCell	FooterCellNamedPart
FooterButtons	FooterButtonsNamedPart
DataRowSeparator	DataRowSeparatorNamedPart
DataRowSeparatorCell	DataRowSeparatorCellNamedPart
NoDataToShow	NoDataToShowNamedPart
GroupContainer	GroupContainerNamedPart
GroupHeader	GroupHeaderNamedPart
GroupFooter	GroupFooterNamedPart
GroupSeparator	GroupSeparatorNamedPart
EmptyItem	EmptyItemNamedPart
EmptyItemCell	EmptyItemCellNamedPart

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

Scaffold Caching Properties

These properties appear in the "Scaffold Caching" category of the Properties Editor.

Automatic Scaffolding can cache the results of collecting a list of DataFieldInPattern objects to avoid generating the same list multiple times. These properties determine if scaffolding is used and how long data is preserved in the cache.

- **EnableScaffoldCache** (enum PeterBlum.DES.DynamicData.CacheLocation) – Determines if caching is used and its location when on.

  The user can turn off caching on an individual request simply by setting this to `None`.

  Cached data is assigned to a group with a named defined in **CacheCategoryName**. Separate cached data is kept by the MetaTable.Name requesting the data and by the data entry mode. The actual cache name combines all of these elements.

  The enumerated type CacheLocation has these values:

  - None - Cache is not used. Scaffolding is requested on each DynamicPattern control.
  - Page - Cache lasts for the page request. This allows many DynamicPatterns that work together to create a single data bound control to share a list of DataFieldInPattern objects. This is the default.
  - `User` - Cache per user using the Session class.
  - Global - Cache using the HttpCache class with a priority and timeout.

- **CacheCategoryName** (string) – Defines a category for this cache. Categories let you have multiple caches, for different situations.

  You might have a different dataset per User's role. This could be the role name.

  It defaults to "".

- **CacheExpirationSeconds** (integer) – Determines that entries cached in the cache expire after this many seconds.

  When in the Global cache, ASP.NET will automatically delete them when no longer needed. When in the User cache, they are only abandoned if requested and over this time limit. Not used when using the Page cache.

  It defaults to 60 (seconds).

- **CacheSlidingExpiration** (Boolean) – When using the Global or User cache, this determines if the expiration time (**CacheExpirationSeconds**) resets each time the item is retrieved from the cache.

  When `true`, expiration time resets each time the item is retrieved.

  When `false`, the expiration time is from the time initially added to the cache.

  It defaults to `false`.

- **GlobalCachePriority** (enum System.Web.Caching.CacheItemPriority) – When using the Global cache, this is the priority to preserve the data when the cache needs more memory.

  It defaults to `CacheItemPriority.Normal`.

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

Behavior Properties

- **Visible** (Boolean) – When `false`, no HTML is output. This control is entirely unused. It defaults to `true`.
- **AutoLoadPatternTemplateStyleSheetFile** (Boolean) – Pattern Templates often use the style sheet classes defined in the file: **~/DynamicData/PatternTemplateNamedStyles.css**. When this property is `true`, that file is automatically added to the page without you establishing a `<link>` tag.

  When used, it merges the style sheet file with other DES files. It also compresses the file, removing comments and much whitespace.

  When not used, you can still load it merged and compressed by calling

  ```
  PeterBlum.DES.Globals.Page.StyleSheetManager.Include(
   PeterBlum.DES.StyleSheetFiles.PatternTemplateNamedStyles)
  ```

  The path to this file can be changed in two ways:

  - In `Application_Start()`, call

    ```
    PeterBlum.DES.StyleSheetManager.SetDefaultUrl(
     PeterBlum.DES.StyleSheetFiles.PatternTemplateNamedStyles, "URL")
    ```

  - In the `<appSettings>` section of **web.config**, add

    ```
    <add key="DES_StyleSheetPatternTemplateNamedStyles" value="URL" />
    ```

  It defaults to `true`.

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

Methods for the DynamicPattern Control

These methods are intended for the Page developer to create and retrieve ItemInPattern objects.

Click on any of these topics to jump to them:

- AddDataField() method
- AddDataFieldList() method
- AddButtons() method
- AddScaffolding() method
- FindByID() method
- FindByDataField() method
- GetAttributeAsInteger() method
- GetAttributeAsBoolean() method
- GetAttributeAsString() method
- GetAttributeAsUnit() method
- GetAttributeAsNative() method
- SetAttributeAsNative() method

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

AddDataField() method

Adds a data field to the end of the **ItemsInPattern** collection. It creates a DataFieldInPattern object for that data field and returns it for further modification.

[C#]

```
PeterBlum.DES.DynamicData.DataFieldInPattern AddDataField(string pDataField)
```

[VB]

```
Function AddDataField(ByVal pDataField As String) _
  As PeterBlum.DES.DynamicData.DataFieldInPattern
```

Parameters

pDataField

The data field name to add.

Returns

The DataFieldInPattern object that was created and added to the end of the collection.

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

AddDataFieldList() method

It creates a DataFieldListInPattern object with the NamedPart passed in and returns it for further modification. Generally you will call the `AddDataField()` method of the DataFieldListInPattern object to add data fields to it.

[C#]

```
PeterBlum.DES.DynamicData.DataFieldListInPattern AddDataFieldList(string pNamedPart)
```

[VB]

```
Function AddDataFieldList(ByVal pNamedPart As String) _
  As PeterBlum.DES.DynamicData.DataFieldListInPattern
```

Parameters

pNamedPart

The name of a NamedPart. It will be assigned to DataFieldListInPattern.NamedPart property.

Returns

The DataFieldListInPattern object that was created and added to the end of the collection.

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

AddButtons() method

It creates a ButtonsInPattern object with the NamedPart passed in and returns it for further modification.

[C#]

```
PeterBlum.DES.DynamicData.ButtonsInPattern AddButtons(string pNamedPart)
```

[VB]

```
Function AddButtons(ByVal pNamedPart As String) _
  As PeterBlum.DES.DynamicData.ButtonsInPattern
```

Parameters

pNamedPart

The name of a NamedPart. It will be assigned to ButtonsInPattern.NamedPart property.

Returns

The ButtonsInPattern object that was created and added to the end of the collection.

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

AddScaffolding() method

Creates a ScaffoldInPattern object at the end of the **ItemsInPattern** collection.

[C#]

```
PeterBlum.DES.DynamicData.ScaffoldInPattern AddScaffolding()
```

```
PeterBlum.DES.DynamicData.ScaffoldInPattern AddScaffolding(
  string pNamedPart)
```

[VB]

```
Function AddScaffolding() As PeterBlum.DES.DynamicData.ScaffoldInPattern
```

```
Function AddScaffolding(ByVal pNamedPart As String) _
  As PeterBlum.DES.DynamicData.ScaffoldInPattern
```

Parameters

pNamedPart

The named of a NamedPart within the PartsPatternTemplate. Pass "" to use standard `<DataRow>`.

Returns

The ScaffoldInPattern object that was created and added to the end of the collection.

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

FindByID() method

Finds any ItemInPattern object with the specified ID within the **ItemsInPattern** collection. Searches child DataFieldListInPattern.ItemsInPattern collections too.

Each ItemInPattern has an **ID** property where you can assign a unique name. Its purpose is to be used with this method to quickly retrieve it.

[C#]

```
PeterBlum.DES.DynamicData.BaseItemInPattern FindByID(string pID)
```

[VB]

```
Function FindByID(ByVal pID As String) _
  As PeterBlum.DES.DynamicData.BaseItemInPattern
```

Parameters

pID

The ID to find amongst the ItemInPattern objects.

Returns

The ItemInPattern object or `null`/`Nothing` if not found. Typecast the result to the expected class type.

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

FindByDataField() method

Finds a DataFieldInPattern object with the specified data field name within the **ItemsInPattern** collection. Searches child DataFieldListInPattern.ItemsInPattern collections too.

[C#]

```
PeterBlum.DES.DynamicData.DataFieldInPattern FindByDataField(string pDataField)
```

[VB]

```
Function FindByDataField(ByVal pDataField As String) _
  As PeterBlum.DES.DynamicData.DataFieldInPattern
```

Parameters

pDataField

The data field name to find amongst the ItemInPattern objects. The search is case insensitive.

Returns

The DataFieldInPattern object or `null`/`Nothing` if not found.

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

GetAttributeAsInteger() method

Gets a value from the Attributes collection (a list of properties defined in the ASP.NET markup that have no equivalent property name on the DynamicControl). If it is not found, return the *pDefaultValue* value.

Use this in Pattern Templates. Declare public properties in the Pattern Template code that call this to retrieve a setting supplied by the user through the DynamicPattern control. See "Passing properties to be used by the PatternTemplate's own code".

This method returns type integer.

[C#]

```
int GetAttributeAsInteger(string pKey, int pDefaultValue)
```

[VB]

```
Function GetAttributeAsInteger(ByVal pKey As String, ByVal pDefaultValue As Integer) _
  As Integer
```

Parameters

pKey

The name for the Attribute. It should be the same text as the property name within the Pattern Template.

pDefaultValue

The value that is returned if the user has not supplied a value for the *pKey* attribute.

Returns

The value from the *pKey* attribute or *pDefaultValue* if pKey was not supplied by the user.

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

GetAttributeAsBoolean() method

Gets a value from the Attributes collection (a list of properties defined in the ASP.NET markup that have no equivalent property name on the DynamicControl). If it is not found, return the *pDefaultValue* value.

Use this in Pattern Templates. Declare public properties in the Pattern Template code that call this to retrieve a setting supplied by the user through the DynamicPattern control. See "Passing properties to be used by the PatternTemplate's own code".

[C#]

```
bool GetAttributeAsBoolean(string pKey, bool pDefaultValue)
```

[VB]

```
Function GetAttributeAsBoolean(ByVal pKey As String, ByVal pDefaultValue As Boolean) _
  As Boolean
```

Parameters

pKey

The name for the Attribute. It should be the same text as the property name within the Pattern Template.

pDefaultValue

The value that is returned if the user has not supplied a value for the *pKey* attribute.

Returns

The value from the *pKey* attribute or *pDefaultValue* if pKey was not supplied by the user.

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

GetAttributeAsString() method

Gets a value from the Attributes collection (a list of properties defined in the ASP.NET markup that have no equivalent property name on the DynamicControl). If it is not found, return the *pDefaultValue* value.

Use this in Pattern Templates. Declare public properties in the Pattern Template code that call this to retrieve a setting supplied by the user through the DynamicPattern control. See "Passing properties to be used by the PatternTemplate's own code".

[C#]

```
string GetAttributeAsString(string pKey, string pDefaultValue)
```

[VB]

```
Function GetAttributeAsString(ByVal pKey As String, ByVal pDefaultValue As String) _
  As String
```

Parameters

pKey

The name for the Attribute. It should be the same text as the property name within the Pattern Template.

pDefaultValue

The value that is returned if the user has not supplied a value for the *pKey* attribute.

Returns

The value from the *pKey* attribute or *pDefaultValue* if pKey was not supplied by the user.

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

GetAttributeAsUnit() method

Gets a value from the Attributes collection (a list of properties defined in the ASP.NET markup that have no equivalent property name on the DynamicControl). If it is not found, return the *pDefaultValue* value.

Use this in Pattern Templates. Declare public properties in the Pattern Template code that call this to retrieve a setting supplied by the user through the DynamicPattern control. See "Passing properties to be used by the PatternTemplate's own code".

[C#]

```
System.Web.UI.WebControls.Unit GetAttributeAsUnit(string pKey,
   System.Web.UI.WebControls.Unit pDefaultValue)
```

[VB]

```
Function GetAttributeAsUnit(ByVal pKey As String, _
   ByVal pDefaultValue As System.Web.UI.WebControls.Unit) _
      As System.Web.UI.WebControls.Unit
```

Parameters

pKey

The name for the Attribute. It should be the same text as the property name within the Pattern Template.

pDefaultValue

The value that is returned if the user has not supplied a value for the *pKey* attribute.

Returns

The value from the *pKey* attribute or *pDefaultValue* if pKey was not supplied by the user.

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

GetAttributeAsNative() method

Gets a value from the Attributes collection (a list of properties defined in the ASP.NET markup that have no equivalent property name on the DynamicControl). If it is not found, return the *pDefaultValue* value.

Use this in Pattern Templates. Declare public properties in the Pattern Template code that call this to retrieve a setting supplied by the user through the DynamicPattern control. See "Passing properties to be used by the PatternTemplate's own code".

This method returns type String. Use it when you don't have an Integer, Boolean, or String type because those are supported by the `GetAttributeAsInteger()`, `GetAttributeAsBoolean()`, and `GetAttributeAsString()` methods.

[C#]

```
String GetAttributeAsNative(string pKey, String pDefaultValue)
```

[VB]

```
Function GetAttributeAsNative(ByVal pKey As String, ByVal pDefaultValue As String) _
  As String
```

Parameters

pKey

The name for the Attribute. It should be the same text as the property name within the Pattern Template.

pDefaultValue

The value that is returned if the user has not supplied a value for the *pKey* attribute.

Returns

The value from the *pKey* attribute or *pDefaultValue* if pKey was not supplied by the user.

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

SetAttributeAsNative() method

Sets the value of a property found in the Pattern Template into the Attributes collection (a list of properties defined in the ASP.NET markup that have no equivalent property name on the DynamicControl).

Use this when your Pattern Template defines a public property whose value you want to modify. See "Passing properties to be used by the PatternTemplate's own code".

[C#]

```
void SetAttributeAsNative(string pKey, object pValue)
```

[VB]

```
Sub SetAttributeAsNative(ByVal pKey As String, ByVal pValue As Object)
```

Parameters

pKey

The name for the Attribute. It should be the same text as the property name within the Pattern Template.

pValue

The value that is returned if the user has not supplied a value for the *pKey* attribute.

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

The DataFieldInPattern Class

The `PeterBlum.DES.DynamicData.DataFieldInPattern` class plays a central role when using Pattern Templates. It holds a single data field name and maps it to a DynamicControl within the Pattern Template. It has these features:

- When the Pattern Template has multiple DynamicControls, you can identify a specific one either by its ID or position amongst other DynamicControls.
- Can pass UIHint and various attributes down to the DynamicControl. UIHint determines which Field Template is used. The other attributes are assigned to properties on the Field Template that was loaded.
- Can map a data field to a DynamicLabel control. It can also supply custom text to the label instead of the actual DisplayName for the data field.
- Can select a NamedPart from a PartsPatternTemplate to provide an alternative UI.

Click on any of these topics to jump to them:

- Using the DataFieldInPattern class
- Properties of the DataFieldInPattern class
- Constructors for the DataFieldInPattern class

```
<desDD:DynamicPattern id="id" runat="server" PatternTemplateName="Name" />
  <ItemsInPattern>
    <desDD:DataFieldInPattern DataField="ColumnName1" />
    <desDD:DataFieldInPattern DataField="ColumnName2" />
    <desDD:DataFieldInPattern DataField="ColumnName3" UIHint="Calendar" />
    <desDD:DataFieldInPattern DataField="ColumnName4" NamedPart="name" />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

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

Using the DataFieldInPattern class

This class is used with Pattern Templates that contain DynamicControls, including those subclassed from DataFieldsPatternTemplate and PartsPatternTemplate. It should **not** be used with the following Pattern Template types: NoDataFieldsPatternTemplate and DynamicButtonsPatternTemplate.

Its primary task is to assign the **DataField** property on a DynamicControl.

Click on any of these topics to jump to them:

- Basic setup: Always assign data field name
- Use a NamedPart in a PartsPatternTemplate
- Specify a UIHint to identify the Field Template
- <desDD:DataFieldInPattern DataField="ColumnName" ReadOnly="True" />
- Supply alternative values for properties on the Field Template
- Customizing the text for a DynamicLabel
- Specify which DynamicControl and DynamicLabel gets assigned the data field

Basic setup: Always assign data field name

Specify the desired data field name in the **DataField** property. Use the name of the property, not any name specified on a DisplayNameAttribute assigned to that column.

```
<desDD:DataFieldInPattern DataField="ColumnName" />
```

The DataFieldInPattern object can be created programmatically by calling the `AddDataField("DataField")` method. Its available on the DynamicPattern, DynamicFormView, and DynamicListView controls and the DataFieldListInPattern object. See "AddDataField() method".

Use a NamedPart in a PartsPatternTemplate

When using a PartsPatternTemplate, you can provide alternatives to the HTML output for each Part. Within the Pattern Template, create a `<NamedParts>` node inside the desired Part. Add a *PartType*NamedPatternPart object with a unique name to the `<NamedParts>` node and use its `<Template>` to define the DynamicControl, HTML, and other controls that make up your alternative user interface.

Here is an example, taken from the **GridView.ascx** file, that shows multiple NamedParts for the <DataCell> Part:

```
<DataCell NamedStyle="DataCell" CssClass="<%#cCssDataCell %>"
      Style="vertical-align:top;" >
   <NamedParts>
      <desDD:DataCellNamedPatternPart Name="TwoStacked">
         <Template>
            <desDD:DynamicControl ID="DynamicControl1" runat="server" /><br />
            <desDD:DynamicControl ID="DynamicControl2" runat="server" />
         </Template>
      </desDD:DataCellNamedPatternPart>
      <desDD:DataCellNamedPatternPart Name="ShowLabel">
         <Template>
            <desDD:DynamicLabel ID="DynamicLabel1" runat="server" />
         </Template>
      </desDD:DataCellNamedPatternPart>
   </NamedParts>
</DataCell>
```

If you need templates that are mode-specific, use <Template>, <EditTemplate> and <InsertTemplate> for read-only, edit, and insert modes respectively. If <EditTemplate> or <InsertTemplate> are not declared, it falls back to using <Template>.

Assign the unique name to the **NamedPart** property on the DataFieldInPattern object.

```
<desDD:DataFieldInPattern DataField="ColumnName"
```

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

```
NamedPart="ShowLabel" />
```

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

Specify a UIHint to identify the Field Template

Use the DataFieldInPattern's **UIHint** property to override the Field Template selected by the metadata. Specify the name of the Field Template, omitting the extensions ".ascx", "_edit.ascx", and "_insert.ascx".

```
<desDD:DataFieldInPattern DataField="ColumnName" UIHint="TemplateName" />
```

You can apply mode-specific UIHints with these properties: **ReadOnlyUIHint**, **EditUIHint**, **InsertUIHint**. **UIHint** is the default for any that is not assigned.

If you need to force the Field Template to its ReadOnly format, such as when security prevents editing the column, set **ReadOnly** to `true`.

```
<desDD:DataFieldInPattern DataField="ColumnName" ReadOnly="True" />
```

Supply alternative values for properties on the Field Template

If you want to set a public property found on a Field Template, specify that property name using one of these properties: **DefaultPropertyList**, **ReadOnlyPropertyList**, **EditPropertyList**, **InsertPropertyList**. See "Field Template Properties".

Each supports the ASP.NET Declarative Syntax for properties.

```
<desDD:DataFieldInPattern DataField="ColumnName"
  EditPropertyList="TextBox-CssClass='UltraTextBox'
       ColumnValidatorManager-ErrorFormatter-CssClass='UltraValidate'" />
```

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

Customizing the text for a DynamicLabel

By default, a DynamicLabel will get the data field's DisplayName. When this text is not correct, you can replace it with the **LabelText**, **HeaderText**, or **FooterText** property. **HeaderText** applies to headers. **FooterText** applies to footers. **LabelText** applies to anything else.

It can show any HTML you want. If you want to include the DisplayName, embed the "{COLUMN}" token. It can also lookup a different column name by using the syntax "{COLUMN:*column name*}"

```
<desDD:DataFieldInPattern DataField="ColumnName" LabelText="Different Label" />

<desDD:DataFieldInPattern DataField="ColumnName" LabelText="{COLUMN}:" />

<desDD:DataFieldInPattern DataField="ColumnName"
  LabelText="From {COLUMN} to {COLUMN:columnname2}" />

<desDD:DataFieldInPattern DataField="ColumnName" HeaderText="Price"
  FooterText="<hr />" />
```

Specify which DynamicControl and DynamicLabel gets assigned the data field

If the Pattern Template has multiple DynamicControls or DynamicLabels, add a list of `DataFieldInPattern` objects to the **ItemsInPattern** property of the DynamicPattern control or a DataFieldListInPattern object. The order of data fields in the list is the order they are applied against the DynamicControls within the Pattern Template.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
  PatternTemplateName="MyPatternTemplate" >
  <ItemsInPattern>
    <desDD:DataFieldInPattern DataField="columnname1" />
    <desDD:DataFieldInPattern DataField="columnname2" />
    <desDD:DataFieldInPattern DataField="columnname3" />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

Use the **ControlIDInPattern** or **PositionInPattern** properties to express the exact DynamicControl or DynamicLabel that receives the data fields. **ControlIDInPattern** specifies by the ID property on the control. **PositionInPattern** specifies by its order, where 0 is the first DynamicControl.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
  PatternTemplateName="MyPatternTemplate" >
  <ItemsInPattern>
    <desDD:DataFieldInPattern DataField="columnname1"
        ControlIDInPattern="ControlID1" />
    <desDD:DataFieldInPattern DataField="columnname2"
        ControlIDInPattern="ControlID2" />
    <desDD:DataFieldInPattern DataField="columnname3"
        ControlIDInPattern="ControlID3" />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

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

Properties of the DataFieldInPattern class

Click on any of these topics to jump to them:

- Data Properties
- Behavior Properties
- Field Template Properties

Data Properties

- **DataField** (string) – The data field name assigned to the DynamicControl or DynamicLabel within the Pattern Template. This is a required property. It defaults to "".

  For more, see "Basic setup: Always assign data field name".

- **ReadOnly** (boolean) – Force the Field Template to be in a ReadOnly mode even if others are in Edit or Insert modes.

  When `true`, the field is readonly.

  It defaults to `false`.

- **LabelText** (string) – Assists in formatting the text assigned to a DynamicLabel control. This property is used except when a header or footer is being generated. When the **DynamicPattern.PartsToGenerate** property indicates a header, use **HeaderText**. When the **DynamicPattern.PartsToGenerate** property indicates a footer, use **FooterText**.

  When not assigned or it has a "{COLUMN}" token, the column's metadata supplies several forms of text available to display: Display Name, Short Display Name, Description and Prompt. They are determined by the DisplayNameAttribute, DescriptionAttribute, and DisplayAttribute on the metadata. The DynamicLabel control within the PatternTemplate selects which of these to use in its TextSource and TextSourceFallback properties.

  When assigned, this text is used instead. It can contain a more suitable text label and HTML tags. It can contain the "{COLUMN}" token to use a name from metadata as described above. It can also lookup a different column name by using the syntax "{COLUMN:*column name*}".

```
<desDD:DataFieldInPattern DataField="ColumnName" LabelText="Different Label" />
<desDD:DataFieldInPattern DataField="ColumnName" LabelText="{COLUMN}:" />
<desDD:DataFieldInPattern DataField="ColumnName"
  LabelText="From {COLUMN} to {COLUMN:columnname2}" />
```

  It determines the data field through the **AssociatedControlID** property on DynamicLabel or the **ControlIDInPattern** property specifies the ID to the DynamicLabel.

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

- **HeaderText** (string) – Assists in formatting the text assigned to a DynamicLabel or DynamicColumnTitle control when that control is in a Header. Use it to change the default text of the column title. The **DynamicPattern.PartsToGenerate** property must indicate a header to use **HeaderText**. If the DataFieldInPattern is declared in a DynamicListView, it will apply this to the DynamicLabel or DynamicColumnTitle in the `<HeaderCell>` Part.

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

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

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.

- **FooterText** (string) – Assists in formatting the text assigned to a DynamicLabel control when that control is in a Footer. Use it to assign the text to the Label. The **DynamicPattern.PartsToGenerate** property must indicate a Footer to use **FooterText**. If the DataFieldInPattern is declared in a DynamicListView, it will apply this to the DynamicLabel or DynamicColumnTitle in the `<FooterCell>` Part.

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

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

Behavior Properties

- **ControlIDInPattern** (string) – When there are multiple DynamicControls, this can be used to identify which is used with this DataFieldInPattern object. Specify the ID value of the DynamicControl here.

  When using DynamicLabels, this DataFieldInPattern can directly set the **DataField** property on the DynamicLabel by setting the ID value of the DynamicLabel control here. Use this capability when the DynamicLabel does not point to another DynamicControl through its **AssociatedControlID** property.

  It defaults to "".

  For more, see "Specify which DynamicControl and DynamicLabel gets assigned the data field".

- **PositionInPattern** (integer) – When there are multiple DynamicControls, this can be used to identify which is used with this DataFieldInPattern object. If 0 or higher, it specifies the position of a DynamicControl (but *not* a DynamicLabel) found in the Pattern's control tree. If -1, it uses the position of this DataFieldInPattern in the **ItemsInPattern** collection to determine the position.

  When **ControlIDInPattern** is used, **PositionInPattern** is always ignored.

  When -1, it uses the position of this DataFieldInPattern in the ItemsInPattern collection to determine the position.

  0 matches the first DynamicControl.

  It defaults to -1.

  Positions are determined within the Pattern Template. It can allow the base class to search its child controls and build a list ordered as they are found in the control tree. For DataFieldsPatternTemplates, look for an overridden `RegisterControls()` method and use the order of calls to `RegisterDynamicControl()` and `RegisterDynamicLabel()` to dictate the position in the list.

  If you want to update both a DynamicLabel and its associated DynamicControl using the same DataFieldInPattern object, be sure to set the **DynamicLabel.AssociatedControlID** property to the DynamicControl within your Pattern Template.

  For more, see "Specify which DynamicControl and DynamicLabel gets assigned the data field".

- **NamedPart** (string) – *PartsPatternTemplate only.*When assigned, it specifies the name of a NamedPart in the NamedParts collection of the Part being used to generate the HTML. When blank, always use the standard formatting for the Part.

  It defaults to "".

  For more, see "Use a NamedPart in a PartsPatternTemplate".

CONTINUED ON THE NEXT PAGE

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

- **ID** (string) – Optional identifier to make it easy to retrieve this object from a collection of ItemInPattern objects.

  It defaults to "".

  To get the object by its ID, call the `FindByID("ID")` method. Its declared on these classes: DynamicPattern, DynamicListView, DynamicFormView, and DataFieldListInPattern.

  Example

  Assign the DataField property on ID="NeedsDataField".

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
  PatternTemplateName="MyPatternTemplate" >
  <ItemsInPattern>
    <desDD:DataFieldInPattern DataField="columnname1" />
    <desDD:DataFieldInPattern ID="NeedsDataField" />
    <desDD:DataFieldInPattern DataField="columnname3" />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

  [C#]

```
BaseItemInPattern vItemInPattern = DynamicPattern1.FindByID("NeedsDataField");
if ((vItemInPattern != null) && (vItemInPattern is DataFieldInPattern))
    ((DataFieldInPattern)vItemInPattern).DataField = "LongDescription";
```

  [VB]

```
Dim vItemInPattern As BaseItemInPattern = _
   DynamicPattern1.FindByID("NeedsDataField")
If (vItemInPattern IsNot Nothing) And _
   (TypeOf vItemInPattern Is DataFieldInPattern) Then
   CType(vItemInPattern, DataFieldInPattern).DataField = "LongDescription"
End If
```

- **Enabled** (Boolean) – Determines if the object is used or ignored. When `true`, it is used. When `false`, it and its children are not used. It defaults to `true`.

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

Field Template Properties

- **UIHint** (string) – The Field Template name requested when no mode specific **UIHint** property is used (below). If blank, use the FieldTemplateFactory's rules to find the Field Template. It defaults to "".
- **ReadOnlyUIHint** (string) – The Field Template name requested for ReadOnly mode. If blank, use the **UIHint** property. It defaults to "".
- **EditUIHint** (string) – The Field Template name requested for Edit mode. If blank, use the **UIHint** property. It defaults to "".
- **InsertUIHint** (string) – The Field Template name requested for Insert mode. If blank, use the **UIHint** property. It defaults to "".
- **DefaultPropertyList** (String) – A list of properties on the Field Template that you want to set on any Field Template. It can be overridden with mode-specific **PropertyList** properties below.

  The list shows `PropertyName='value'` in the ASP.NET Declarative Syntax. If you need a single quote within the value, use `&#39` for that character.

  Property names support nested properties using the syntax `PropertyName-ChildPropertyName`.

  It defaults to "".

```
<desDD:DataFieldInPattern DataField="ProductName"
  DefaultPropertyList="Width='200px' TextBox-CssClass='UltraTextBox'" />
```

- **ReadOnlyPropertyList** (string) – A list of properties on the Field Template that you want to set on the Field Template in ReadOnly mode.

  The list shows `PropertyName='value'` in the ASP.NET Declarative Syntax. If you need a single quote within the value, use `&#39` for that character.

  Property names support nested properties using the syntax `PropertyName-ChildPropertyName`.

  It defaults to "".

```
<desDD:DataFieldInPattern DataField="ProductName"
  ReadOnlyPropertyList="Width='200px' DataControl-CssClass='UltraLabel'" />
```

- **EditPropertyList** (string) – A list of properties on the Field Template that you want to set on the Field Template in Edit mode.

  The list shows `PropertyName='value'` in the ASP.NET Declarative Syntax. If you need a single quote within the value, use `&#39` for that character.

  Property names support nested properties using the syntax `PropertyName-ChildPropertyName`.

  It defaults to "".

```
<desDD:DataFieldInPattern DataField="ProductName"
  EditPropertyList="Width='200px' TextBox-CssClass='UltraTextBox'" />
```

- **InsertPropertyList** (string) – A list of properties on the Field Template that you want to set on the Field Template in Edit mode.

  The list shows `PropertyName='value'` in the ASP.NET Declarative Syntax. If you need a single quote within the value, use `&#39` for that character.

  Property names support nested properties using the syntax `PropertyName-ChildPropertyName`.

  It defaults to "".

```
<desDD:DataFieldInPattern DataField="ProductName"
  EditPropertyList="Width='200px' TextBox-CssClass='UltraTextBox'" />
```

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

Constructors for the DataFieldInPattern class

These constructor declarations refer to the property names they set.

[C#]

```
DataFieldInPattern()

DataFieldInPattern(string pDataField)

DataFieldInPattern(string pDataField, string pControlIDInPattern)

DataFieldInPattern(string pDataField, string pControlIDInPattern,
         string pUIHint)

DataFieldInPattern(string pDataField, string pControlIDInPattern,
         string pReadOnlyUIHint, string pEditUIHint, string pInsertUIHint)

DataFieldInPattern(string pDataField, string pControlIDInPattern,
         string pReadOnlyUIHint, string pEditUIHint, string pInsertUIHint,
         string pReadOnlyPropertyList, string pEditPropertyList,
         string pInsertPropertyList)

DataFieldInPattern(string pDataField, int pPositionInPattern)

DataFieldInPattern(string pDataField, int pPositionInPattern,
         string pUIHint)

DataFieldInPattern(string pDataField, int pPositionInPattern,
         string pReadOnlyUIHint, string pEditUIHint, string pInsertUIHint)

DataFieldInPattern(string pDataField, int pPositionInPattern,
         string pReadOnlyUIHint, string pEditUIHint, string pInsertUIHint,
         string pReadOnlyPropertyList, string pEditPropertyList,
         string pInsertPropertyList)
```

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

[VB]

```
New()

New(ByVal pDataField As String)

New(ByVal pDataField As String, ByVal pControlIDInPattern As String)

New(ByVal pDataField As String, ByVal pControlIDInPattern As String,
         ByVal pUIHint As String)

New(ByVal pDataField As String, ByVal pControlIDInPattern As String,
         ByVal pReadOnlyUIHint As String, ByVal pEditUIHint As String,
         ByVal pInsertUIHint As String)

New(ByVal pDataField As String, ByVal pControlIDInPattern As String,
         ByVal pReadOnlyUIHint As String, ByVal pEditUIHint As String,
         ByVal pInsertUIHint As String,
         ByVal pReadOnlyPropertyList As String, ByVal pEditPropertyList As String,
         ByVal pInsertPropertyList As String)

New(ByVal pDataField As String, ByVal pPositionInPattern As Integer)

New(ByVal pDataField As String, ByVal pPositionInPattern As Integer,
         ByVal pUIHint As String)

New(ByVal pDataField As String, ByVal pPositionInPattern As Integer,
         ByVal pReadOnlyUIHint As String, ByVal pEditUIHint As String,
         ByVal pInsertUIHint As String)

New(ByVal pDataField As String, ByVal pPositionInPattern As Integer,
         ByVal pReadOnlyUIHint As String, ByVal pEditUIHint As String,
         ByVal pInsertUIHint As String,
         ByVal pReadOnlyPropertyList As String, ByVal pEditPropertyList As String,
         ByVal pInsertPropertyList As String)
```

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

The DataFieldListInPattern Class

When a NamedPart contains two or more DynamicControls, use the `PeterBlum.DES.DynamicData.DataFieldListInPattern` object to supply the list of data fields to that Part. DataFieldListInPattern is only used with a PartsPatternTemplate.

Click on any of these topics to jump to them:

- Using the DataFieldListInPattern Class
- Properties of the DataFieldListInPattern class
- Methods for the DataFieldListInPattern class
- Constructors for the DataFieldListInPattern class


```
<desDD:DynamicPattern id="id" runat="server" PatternTemplateName="Name" />
  <ItemsInPattern>
    <desDD:DataFieldListInPattern NamedPart="NamedPart" />
    <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="ColumnName1" />
      <desDD:DataFieldInPattern DataField="ColumnName2" />
      <desDD:DataFieldInPattern DataField="ColumnName3" />
    </ItemsInPattern>
    </desDD:DataFieldListInPattern>
  </ItemsInPattern>
</desDD:DynamicPattern>
```

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

Using the DataFieldListInPattern Class

When using a PartsPatternTemplate, you can create a list of "NamedParts" for each Part. NamedParts have an identifying name and a template containing any web controls and HTML. They are used when you specify the **NamedPart** property on any ItemInPattern object as an alternative to standard formatting for the Part.

Here is an example taken from the **GridView.ascx** Pattern Template.

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server" >

   <Various Parts>

<DataCell NamedStyle="DataCell" CssClass="<%#cCssDataCell %>"
      Style="vertical-align:top;" >
   <NamedParts>
      <desDD:DataCellNamedPatternPart Name="TwoStacked">
         <Template>
            <desDD:DynamicControl ID="DynamicControl1" runat="server" /><br />
            <desDD:DynamicControl ID="DynamicControl2" runat="server" />
         </Template>
      </desDD:DataCellNamedPatternPart>
      <desDD:DataCellNamedPatternPart Name="ShowLabel">
         <Template>
            <desDD:DynamicLabel ID="DynamicLabel1" runat="server" />
         </Template>
      </desDD:DataCellNamedPatternPart>
   </NamedParts>
</DataCell>

</desDD:DynamicPartsPatterns>
```

If you need templates that are mode-specific, use `<Template>`, `<EditTemplate>` and `<InsertTemplate>` for read-only, edit, and insert modes respectively. If `<EditTemplate>` or `<InsertTemplate>` are not declared, it falls back to using `<Template>`.

The `PeterBlum.DES.DynamicData.DataFieldListInPattern` class is used when the NamedPart contains two or more DynamicControls. Specify the NamedPart name within the **NamedPart** property. Populate its **ItemsInPattern** collection with DataFieldInPattern objects, one for each DynamicControl.

```
<desDD:DataFieldListInPattern NamedPart="Name" >
  <ItemsInPattern>
    <desDD:DataFieldInPattern DataField="ColumnName1" />
    <desDD:DataFieldInPattern DataField="ColumnName2" />
  </ItemsInPattern>
</desDD:DataFieldListInPattern>
```

DataFieldInPattern objects can be added to its ItemsInPattern collection programmatically by calling the AddDataField() method.

The DataFieldListInPattern is also used when the standard formatting for a Part has more than one DynamicControl. In that case, do not specify a value in the **NamedPart** property

```
<desDD:DataFieldListInPattern>
  <ItemsInPattern>
    <desDD:DataFieldInPattern DataField="ColumnName1" />
    <desDD:DataFieldInPattern DataField="ColumnName2" />
  </ItemsInPattern>
</desDD:DataFieldListInPattern>
```

The DataFieldInPattern objects have a number of their own properties, to define the specific DynamicControl, or perhaps a DynamicLabel and to setup the Field Template. See "Using the DataFieldInPattern class". Do not use their own **NamedPart** property though.

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

The DataFieldListInPattern object can be created programmatically by calling the `AddDataFieldList("templatename")` method. Its available on the DynamicPattern, DynamicFormView, and DynamicListView controls. See “AddDataFieldList() method”.

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

Properties of the DataFieldListInPattern class

Click on any of these topics to jump to them:

- Key Properties
- Behavior Properties

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

Key Properties

- **NamedPart** (string) – *PartsPatternTemplate only.*When assigned, it specifies the name of a NamedPart in the NamedParts collection of the Part being used to generate the HTML. When blank, always use the standard formatting for the Part.

  It defaults to "".

  For more, see "Using the DataFieldListInPattern Class".

- **ItemsInPattern** (PeterBlum.DES.DynamicData.ItemsInPattern) – Populate this list with DataFieldInPattern objects, one for each DynamicControl within the template. You can also call the `AddDataField("DataField")` method on the DataFieldListInPattern class. See "AddDataField() method"'.

  The DataFieldInPattern objects have a number of their own properties, to define the specific DynamicControl, or perhaps a DynamicLabel and to setup the Field Template. Do not use their own **NamedPart** property though.

  See "Using the DataFieldInPattern class".

  If you need to programmatically find a specific object within **ItemsInPattern**, there are two methods available on the DynamicPattern control:

  - For DataFieldInPattern objects, use `FindByDataField("datafieldname")`. See "FindByDataField() method".
  - For all objects, assign a unique value in the ItemInPattern object's ID property when the object is first created or in the ASP.NET markup. Then use `FindByID("id")` and typecast to the expected class. See "FindByID() method".

Example: Using a NamedPart

```
<desDD:DataFieldListInPattern NamedPart="TwoStacked" >
   <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="ProductName" />
      <desDD:DataFieldInPattern DataField="Description" />
   </ItemsInPattern>
</desDD:DataFieldListInPattern>
```

Example: Adding to a DynamicPattern programmatically

[C#]

```
using DES=PeterBlum.DES.DynamicData;
…
DES.DataFieldInListPattern vList = new DES.DataFieldInListPattern("TwoStacked");
DES.DataFieldInPattern vDF = new DES.DataFieldInPattern("ProductName");
vDF.UIHint = "ImportantText";
vList.ItemsInPattern.Add(vDF);
vList.AddDataField("Description");  // shortcut
DynamicPattern1.ItemsInPattern.Add(vList);
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
…
Dim vList As DES.DataFieldInListPattern =
   New DES.DataFieldInListPattern("TwoStacked")
Dim vDF As DES.DataFieldInPattern = New DES.DataFieldInPattern("ProductName")
vDF.UIHint = "ImportantText"
vList.ItemsInPattern.Add(vDF)
vList.AddDataField("Description")   ' shortcut
DynamicPattern1.ItemsInPattern.Add(vList)
```

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

Behavior Properties

- **ID** (string) – Optional identifier to make it easy to retrieve this object from a collection of ItemInPattern objects.

  It defaults to "".

  To get the object by its ID, call the `FindByID("ID")` method. Its declared on these classes: DynamicPattern, DynamicListView, DynamicFormView, and DataFieldListInPattern.

  Example

  Assign the DataField property on ID="NeedsDataField".

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
  PatternTemplateName="MyPatternTemplate" >
  <ItemsInPattern>
    <desDD:DataFieldInPattern DataField="columnname1" />
    <desDD:DataFieldInPattern ID="NeedsDataField" />
    <desDD:DataFieldInPattern DataField="columnname3" />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

  [C#]

```
BaseItemInPattern vItemInPattern = DynamicPattern1.FindByID("NeedsDataField");
if ((vItemInPattern != null) && (vItemInPattern is DataFieldInPattern))
    ((DataFieldInPattern)vItemInPattern).DataField = "LongDescription";
```

  [VB]

```
Dim vItemInPattern As BaseItemInPattern = _
   DynamicPattern1.FindByID("NeedsDataField")
If (vItemInPattern IsNot Nothing) And _
   (TypeOf vItemInPattern Is DataFieldInPattern) Then
   CType(vItemInPattern, DataFieldInPattern).DataField = "LongDescription"
End If
```

- **Enabled** (Boolean) – Determines if the object is used or ignored. When `true`, it is used. When `false`, it and its children are not used. It defaults to `true`.

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

Methods for the DataFieldListInPattern class

These methods are intended for the Page developer to create and retrieve ItemInPattern objects.

Click on any of these topics to jump to them:

- AddDataField() method
- AddButtons() method
- FindByID() method
- FindByDataField() method

AddDataField() method

Adds a data field to the end of the **ItemsInPattern** collection. It creates a DataFieldInPattern object for that data field and returns it for further modification.

[C#]

```
PeterBlum.DES.DynamicData.DataFieldInPattern AddDataField(string pDataField)
```

[VB]

```
Function AddDataField(ByVal pDataField As String) _
  As PeterBlum.DES.DynamicData.DataFieldInPattern
```

Parameters

pDataField

The data field name to add.

Returns

The DataFieldInPattern object that was created and added to the end of the collection.

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

AddButtons() method

It creates a ButtonsInPattern object with the NamedPart passed in and returns it for further modification.

[C#]

```
PeterBlum.DES.DynamicData.ButtonsInPattern AddButtons(string pNamedPart)
```

[VB]

```
Function AddButtons(ByVal pNamedPart As String) _
  As PeterBlum.DES.DynamicData.ButtonsInPattern
```

Parameters

pNamedPart

The name of a NamedPart. It will be assigned to ButtonsInPattern.NamedPart property.

Returns

The ButtonsInPattern object that was created and added to the end of the collection.

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

FindByID() method

Finds any ItemInPattern object with the specified ID within the **ItemsInPattern** collection. Searches child DataFieldListInPattern.ItemsInPattern collections too.

Each ItemInPattern has an **ID** property where you can assign a unique name. Its purpose is to be used with this method to quickly retrieve it.

[C#]

```
PeterBlum.DES.DynamicData.BaseItemInPattern FindByID(string pID)
```

[VB]

```
Function FindByID(ByVal pID As String) _
  As PeterBlum.DES.DynamicData.BaseItemInPattern
```

Parameters

pID

The ID to find amongst the ItemInPattern objects.

Returns

The ItemInPattern object or `null`/`Nothing` if not found. Typecast the result to the expected class type.

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

FindByDataField() method

Finds a DataFieldInPattern object with the specified data field name within the **ItemsInPattern** collection. Searches child DataFieldListInPattern.ItemsInPattern collections too.

[C#]

```
PeterBlum.DES.DynamicData.DataFieldInPattern FindByDataField(string pDataField)
```

[VB]

```
Function FindByDataField(ByVal pDataField As String) _
  As PeterBlum.DES.DynamicData.DataFieldInPattern
```

Parameters

pDataField

The data field name to find amongst the ItemInPattern objects. The search is case insensitive.

Returns

The DataFieldInPattern object or `null`/`Nothing` if not found.

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

Constructors for the DataFieldListInPattern class

These constructor declarations refer to the property names they set.

[C#]

```
DataFieldListInPattern()
DataFieldListInPattern(string pNamedPart)
```

[VB]

```
New()
New(ByVal pNamedPart As String)
```

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

The ScaffoldInPattern Class

The `PeterBlum.DES.DynamicData.ScaffoldInPattern` creates a list of data fields to assign to the DynamicControls based on the Automatic Scaffolding feature. This is used when of the **AutoScaffoldDataFields** property is `False` or `WhenEmpty` and **ItemsInPattern** contains objects and you still need automatic scaffolding.

Click on any of these topics to jump to them:

- Using the ScaffoldInPattern class
- Properties of the ScaffoldInPattern class
- Constructors for the ScaffoldInPattern class


```
<desDD:DynamicPattern id="id" runat="server" PatternTemplateName="Name" />
  <ItemsInPattern>
    <desDD:ScaffoldInPattern />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

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

Using the ScaffoldInPattern class

The `PeterBlum.DES.DynamicData.ScaffoldInPattern` object create a list of Data Fields using Automatic Scaffolding. It internally creates DataFieldInPattern objects at its position within the **ItemsInPattern** collection. This class is recommended only for PartsPatternTemplates because they have a variable number of data fields.

Just specify the object itself, without properties, to get a list of fields from Automatic Scaffolding.

```
<desDD:ScaffoldInPattern />
```

The ScaffoldInPattern object can specify a NamedPart in its **NamedPart** property when using a PartsPatternTemplate. This value is copied to the **NamedPart** property on each DataFieldInPattern object that it creates. It refers to the DataCell, HeaderCell, or FooterCell of the row associated with the data fields.

```
<desDD:ScaffoldInPattern NamedPart="Name" />
```

Note: The ScaffoldInPattern object makes for quick setup but cannot customize the Field Templates for individual data fields. If you need to use the UIHint property or set any properties on the Field Template, switch to using DataFieldInPattern objects.

The ScaffoldInPattern object can be created programmatically by calling the `AddScaffolding()` method. Its available on the DynamicPattern, DynamicFormView, and DynamicListView controls. See “AddScaffolding() method”.

The field list generated by Automatic Scaffolding can be cached to avoid regenerating the same field list every time. DynamicPattern, DynamicFormView, and DynamicListView controls all offer Scaffold Caching properties. See “Scaffold Caching Properties”.

Your DataBoundControlAdapter, which is defined in the DynamicDataManager control, hosts many settings to customize the scaffolding process in its **ScaffoldingSettings** property. They include whether the ScaffoldColumnAttribute is involved and using the CategoryAttribute to determine which columns are included. See “Automatic Scaffolding Properties”.

```
<desDD:DynamicDataManager ID="DynamicDataManager1" runat="server"
   AutoLoadForeignKeys="true" >
   <Adapters>
      <desDD:DynamicFormViewAdapter DataBoundControlID="RecordDetails"
         ValidationSummaryControlID="ValidationSummary1"
         SupportsEditMode="true" SupportsInsertMode="false" >
         <ScaffoldingSettings Categories="Important"
            ShowReadOnlyColumnsInEditMode="true" />
      </desDD:DynamicFormViewAdapter>
   </Adapters>
</desDD:DynamicDataManager>
```

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

Properties of the ScaffoldInPattern class

- **NamedPart** (string) – *PartsPatternTemplate only.*When assigned, it specifies the name of a NamedPart in the NamedParts collection of the Part being used to generate the HTML. When blank, always use the standard formatting for the Part.

  It defaults to "".

- **ID** (string) – Optional identifier to make it easy to retrieve this object from a collection of ItemInPattern objects.

  It defaults to "".

  To get the object by its ID, call the `FindByID("ID")` method. Its declared on these classes: DynamicPattern, DynamicListView, DynamicFormView, and DataFieldListInPattern.

  Example

  Assign the **NameItemTemplate** property on ID="NeedsName".

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
  PatternTemplateName="MyPatternTemplate" >
  <ItemsInPattern>
    <desDD:ScaffoldInPattern ID="NeedsName" />
    <desDD:ButtonsInPattern NamedPart="Buttons" />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

  [C#]

```
BaseItemInPattern vItemInPattern = DynamicPattern1.FindByID("NeedsName");
if ((vItemInPattern != null) && (vItemInPattern is ScaffoldInPattern))
    ((ScaffoldInPattern)vItemInPattern).NamedPart = "Fancy";
```

  [VB]

```
Dim vItemInPattern As BaseItemInPattern = _
   DynamicPattern1.FindByID("NeedsName")
If (vItemInPattern IsNot Nothing) And _
   (TypeOf vItemInPattern Is ScaffoldInPattern) Then
   CType(vItemInPattern, ScaffoldInPattern).NamedPart = "Fancy"
End If
```

- **Enabled** (Boolean) – Determines if the object is used or ignored. When `true`, it is used. When `false`, it and its children are not used. It defaults to `true`.

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

Constructors for the ScaffoldInPattern class

These constructor declarations refer to the property names they set.

[C#]

```
ScaffoldInPattern()
ScaffoldInPattern(string pNamedPart)
```

[VB]

```
New()
New(ByVal pNamedPart As String)
```

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

The ButtonsInPattern Class

The `PeterBlum.DES.DynamicData.ButtonsInPattern` is used with PartsPatternTemplates to add a NamedPart that provides buttons. It can provide titles to the DynamicColumnTitle and DynamicLabel controls in the header and footer rows through its **HeaderText**, and **FooterText** properties. The Data Row element should have no DynamicControls or DynamicLabel controls because they will not get their DataField properties assigned.

The **GridView.ascx**, **ListOfDetailViews.ascx**, and **DetailsView.ascx** Pattern Templates declare two NamedParts for buttons, called "Buttons" and "LinkButtons". You use ButtonsInPattern objects with the **NamedPart** property set to either "Buttons" or "LinkButtons".

Click on any of these topics to jump to them:

- Using the ButtonsInPattern class
- Properties of the ButtonsInPattern class
- Constructors for the ButtonsInPattern class

```
<desDD:DynamicPattern id="id" runat="server" PatternTemplateName="GridView" />
  <ItemsInPattern>
    <desDD:ScaffoldInPattern />
    <desDD:ButtonsInPattern NamedPart="Buttons" />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

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

Using the ButtonsInPattern class

The ButtonsInPattern class should only be used with PartsPatternTemplates to apply a NamedPart declared in the Buttons, HeaderButton, and FooterButton Parts.

Each ButtonsInPattern object must specify the NamedPart name in the **NamedPart** property.

```
<desDD:ButtonsInPattern NamedPart="Name" />
```

The **GridView.ascx**, **ListOfDetailViews.ascx**, and **DetailsView.ascx** Pattern Templates can specify these NamedParts to establish buttons: "Buttons" and "LinkButtons".

```
<desDD:ButtonsInPattern NamedPart="Buttons" />
```

```
<desDD:ButtonsInPattern NamedPart="LinkButtons" />
```

GridView.ascx, **ListOfDetailViews.ascx**, and **DetailsView.ascx** internally refer to other Pattern Templates that use the DynamicButtons control to create their buttons. They are **ButtonsForList.ascx**, **LinkButtonsForList.ascx**, **DetailsViewButtons.ascx**, and **DetailsViewLinkButtons.ascx**.

If you create a new Pattern Template file that supports buttons (such as ImageButtons), add a NamedPart to the **GridView.ascx**, **ListOfDetailViews.ascx**, or **DetailsView.ascx** Pattern Template file that to reference your new Pattern Template file. For example:

```
<Buttons>
  <desDD:ButtonsNamedPatternPart Name="ImageButtons"
    PatternTemplateName="ImageButtonsForList" />
</Buttons>
```

With the above defined, you can use this:

```
<desDD:ButtonsInPattern NamedPart="ImageButtons" />
```

Pattern Templates often expose properties to allow customization. See "Passing properties to be used by the PatternTemplate's own code". You can specify these properties and desired values directly in the ButtonsInPattern object as its own properties. (This even works for DynamicPattern control that is nested within the NamedPart, as shown with the ImageButtons example.)

For example, any Pattern Template that uses DynamicButtonsPatternTemplate (like the ImageButtonsForList file proposed above) has a **ShowSelectButton** property which takes a boolean value. Here's how to declare it:

```
<desDD:ButtonsInPattern NamedPart="ImageButtons"
  ShowSelectButton="True" />
```

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

Properties of the ButtonsInPattern class

- **NamedPart** (string) – *PartsPatternTemplate only.*When assigned, it specifies the name of a NamedPart in the NamedParts collection of the Part being used to generate the HTML. When blank, always use the standard formatting for the Part.

  It defaults to "".

- **ID** (string) – Optional identifier to make it easy to retrieve this object from a collection of ItemInPattern objects.

  It defaults to "".

  To get the object by its ID, call the `FindByID("ID")` method. Its declared on these classes: DynamicPattern, DynamicListView, DynamicFormView, and DataFieldListInPattern.

  Example

  Assign the **NameItemTemplate** property on ID="NeedsName".

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
  PatternTemplateName="MyPatternTemplate" >
  <ItemsInPattern>
    <desDD:DataFieldInPattern DataField="columnname1" />
    <desDD:ButtonsInPattern ID="NeedsName" />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

  [C#]

```
BaseItemInPattern vItemInPattern = DynamicPattern1.FindByID("NeedsName");
if ((vItemInPattern != null) && (vItemInPattern is ButtonsInPattern))
    ((ButtonsInPattern)vItemInPattern).NamedPart = "Buttons";
```

  [VB]

```
Dim vItemInPattern As BaseItemInPattern = _
   DynamicPattern1.FindByID("NeedsName")
If (vItemInPattern IsNot Nothing) And _
   (TypeOf vItemInPattern Is ButtonsInPattern) Then
   CType(vItemInPattern, ButtonsInPattern).NamedPart = _
      "Buttons"
End If
```

- **Enabled** (Boolean) – Determines if the object is used or ignored. When `true`, it is used. When `false`, it and its children are not used. It defaults to `true`.

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

Constructors for the ButtonsInPattern class

These constructor declarations refer to the property names they set.

[C#]

```
ButtonsInPattern()
ButtonsInPattern(string pNamedPart)
```

[VB]

```
New()
New(ByVal pNamedPart As String)
```

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

The LabelInPattern Class

The `PeterBlum.DES.DynamicData.LabelInPattern` is used to assign text or HTML to a DynamicLabel. It is one way to customize the content that is output.

Click on any of these topics to jump to them:

- Using the LabelInPattern class
- Properties of the LabelInPattern class
- Constructors for the LabelInPattern class


```
<desDD:DynamicPattern id="id" runat="server" PatternTemplateName="GridView" />
  <ItemsInPattern>
    <desDD:ScaffoldInPattern />
    <desDD:LabelInPattern NamedPart="Title" LabelText="Welcome"
      ControlIDInPattern="DynamicLabel2" />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

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

Using the LabelInPattern class

The LabelInPattern class must assign text to a DynamicLabel. Set the text in its **LabelText** property.

When the LabelInPattern may be applied to a header or footer, use its **HeaderText** or **FooterText** property to assign text to DynamicLabels within the header or footer. When using a DynamicListView control, a single LabelInPattern may be applied to several parts, including the header, data row, and footer. You can pass different text to each by using all 3 properties:

```
<desDD:LabelInPattern LabelText="Welcome" LabelText="For data row"
   HeaderText="For header row" FooterText="For footer row" />
```

If there are multiple DynamicLabels or any DynamicControls, specify the control ID of the DynamicLabel in the **ControlIDInPattern** property or the position amongst the controls with the **PositionInPattern** property.

```
<desDD:LabelInPattern LabelText="Welcome" ControlIDInPattern="DynamicLabel2" />
```

Each LabelInPattern object can specify a NamedPart name in the **NamedPart** property.

```
<desDD:LabelInPattern NamedPart="Name" LabelText="Welcome"
   ControlIDInPattern="DynamicLabel2" />
```

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

Properties of the LabelInPattern class

- **LabelText** (string) – Assists in formatting the text assigned to a DynamicLabel control. This property is used except when a header or footer is being generated. When the **DynamicPattern.PartsToGenerate** property indicates a header, use **HeaderText**. When the **DynamicPattern.PartsToGenerate** property indicates a footer, use **FooterText**.
- **LabelTextLookupID** (string) – Gets the value for **LabelText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **LabelText** 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.
- **HeaderText** (string) – Assists in formatting the text assigned to a DynamicLabel control when that control is in a Header. The **DynamicPattern.PartsToGenerate** property must indicate a header to use **HeaderText**. If the LabelInPattern is declared in a DynamicListView, it will apply this to the DynamicLabel in the `<HeaderCell>` Part.
- **HeaderTextLookupID** (string) – Gets the value for **HeaderText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **HeaderText** 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.
- **FooterText** (string) – Assists in formatting the text assigned to a DynamicLabel control when that control is in a Footer. The **DynamicPattern.PartsToGenerate** property must indicate a Footer to use **FooterText**. If the LabelInPattern is declared in a DynamicListView, it will apply this to the DynamicLabel in the `<FooterCell>` Part.
- **FooterTextLookupID** (string) – Gets the value for **FooterText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **FooterText** 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.
- **ControlIDInPattern** (string) – When there are multiple DynamicLabels and any DynamicControls, this can be used to identify which is used with this LabelInPattern object. Specify the ID value of the DynamicLabel control here.

  It defaults to "".

  For more, see "Specify which DynamicControl and DynamicLabel gets assigned the data field".
- **PositionInPattern** (integer) – When there are multiple DynamicLabels and any DynamicControls, this can be used to identify which is used with this LabelInPattern object. If 0 or higher, it specifies the position of a DynamicLabel found in the Pattern's control tree. If -1, it uses the position of this LabelInPattern in the **ItemsInPattern** collection to determine the position.

  When **ControlIDInPattern** is used, **PositionInPattern** is always ignored.

  When -1, it uses the position of this LabelInPattern in the **ItemsInPattern** collection to determine the position.

  0 matches the first DynamicControl.

  It defaults to -1.

  Positions are determined within the Pattern Template. It can allow the base class to search its child controls and build a list ordered as they are found in the control tree. For DataFieldsPatternTemplates, look for an overridden

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

`RegisterControls()` method and use the order of calls to `RegisterDynamicControl()` and `RegisterDynamicLabel()` to dictate the position in the list.

For more, see "Specify which DynamicControl and DynamicLabel gets assigned the data field".

- **NamedPart** (string) – *PartsPatternTemplate only.*When assigned, it specifies the name of a NamedPart in the NamedParts collection of the Part being used to generate the HTML. When blank, always use the standard formatting for the Part.

  It defaults to "".

- **ID** (string) – Optional identifier to make it easy to retrieve this object from a collection of ItemInPattern objects.

  It defaults to "".

  To get the object by its ID, call the `FindByID("ID")` method. Its declared on these classes: DynamicPattern, DynamicListView, DynamicFormView, and DataFieldListInPattern.

Example

Assign the **NameItemTemplate** property on ID="NeedsName".

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
  PatternTemplateName="MyPatternTemplate" >
  <ItemsInPattern>
    <desDD:DataFieldInPattern DataField="columnname1" />
    <desDD:LabelInPattern ID="NeedsName" />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

[C#]

```
BaseItemInPattern vItemInPattern = DynamicPattern1.FindByID("NeedsName");
if ((vItemInPattern != null) && (vItemInPattern is LabelInPattern))
    ((LabelInPattern)vItemInPattern).NamedPart = "Buttons";
```

[VB]

```
Dim vItemInPattern As BaseItemInPattern = _
   DynamicPattern1.FindByID("NeedsName")
If (vItemInPattern IsNot Nothing) And _
   (TypeOf vItemInPattern Is LabelInPattern) Then
   CType(vItemInPattern, LabelInPattern).NamedPart = _
      "Buttons"
End If
```

- **Enabled** (Boolean) – Determines if the object is used or ignored. When `true`, it is used. When `false`, it and its children are not used. It defaults to `true`.

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

Constructors for the LabelInPattern class

These constructor declarations refer to the property names they set.

[C#]

```
LabelInPattern()
LabelInPattern(string pNamedPart)
```

[VB]

```
New()
New(ByVal pNamedPart As String)
```

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

The NoDataFieldsInPattern Class

The `PeterBlum.DES.DynamicData.NoDataFieldsInPattern` is used with PartsPatternTemplates to add a NamedPart that does not support data fields. The NamedPart should have no DynamicControls or DynamicLabel controls. These NamedParts can insert other types of HTML formatting and web controls. Buttons and labels are a good example; yet you should use ButtonsInPattern and LabelInPattern for them. Instead, consider this for a NamedPart that offers static content, like an image and HTML.

Click on any of these topics to jump to them:

- Using the NoDataFieldsInPattern class
- Properties of the NoDataFieldsInPattern class
- Constructors for the NoDataFieldsInPattern class


```
<desDD:DynamicPattern id="id" runat="server" PatternTemplateName="GridView" />
  <ItemsInPattern>
    <desDD:ScaffoldInPattern />
    <desDD:NoDataFieldsInPattern NamedPart="Graphics" />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

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

Using the NoDataFieldsInPattern class

The NoDataFieldsInPattern class should only be used with PartsPatternTemplates to apply a NamedPart declared in that Pattern Template.

Each NoDataFieldsInPattern object must specify the NamedPart name in the **NamedPart** property.

```
<desDD:NoDataFieldsInPattern NamedPart="Name" />
```

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

Properties of the NoDataFieldsInPattern class

- **NamedPart** (string) – *PartsPatternTemplate only.*When assigned, it specifies the name of a NamedPart in the NamedParts collection of the Part being used to generate the HTML. When blank, always use the standard formatting for the Part.

  It defaults to "".

- **ID** (string) – Optional identifier to make it easy to retrieve this object from a collection of ItemInPattern objects.

  It defaults to "".

  To get the object by its ID, call the `FindByID("ID")` method. Its declared on these classes: DynamicPattern, DynamicListView, DynamicFormView, and DataFieldListInPattern.

  Example

  Assign the **NameItemTemplate** property on ID="NeedsName".

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
  PatternTemplateName="MyPatternTemplate" >
  <ItemsInPattern>
    <desDD:DataFieldInPattern DataField="columnname1" />
    <desDD:NoDataFieldsInPattern ID="NeedsName" />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

  [C#]

```
BaseItemInPattern vItemInPattern = DynamicPattern1.FindByID("NeedsName");
if ((vItemInPattern != null) && (vItemInPattern is NoDataFieldsInPattern))
    ((NoDataFieldsInPattern)vItemInPattern).NamedPart = "Buttons";
```

  [VB]

```
Dim vItemInPattern As BaseItemInPattern = _
   DynamicPattern1.FindByID("NeedsName")
If (vItemInPattern IsNot Nothing) And _
   (TypeOf vItemInPattern Is NoDataFieldsInPattern) Then
   CType(vItemInPattern, NoDataFieldsInPattern).NamedPart = _
      "Buttons"
End If
```

- **Enabled** (Boolean) – Determines if the object is used or ignored. When `true`, it is used. When `false`, it and its children are not used. It defaults to `true`.

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

Constructors for the NoDataFieldsInPattern class

These constructor declarations refer to the property names they set.

[C#]

```
NoDataFieldsInPattern()
NoDataFieldsInPattern(string pNamedPart)
```

[VB]

```
New()
New(ByVal pNamedPart As String)
```

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

The TemplateNotInPattern Class

The `PeterBlum.DES.DynamicData.TemplateNotInPattern` lets you insert HTML, web controls and DynamicControls that do not exist in the Pattern Template. It is a great tool for further customizing your output without changing the Pattern Template in use.

You can either supply a template, with its web controls support, in its **Template** property or just an HTML string within the **HTML** property. Its template can contain DynamicControls and DynamicLabel controls so long as their **DataField** properties are assigned.

When used in a DynamicListView, it may need different output for header or footer rows. TemplateNotInPattern supplies properties for those cases too: **HeaderTemplate**, **HeaderHTML**, **FooterTemplate**, and **FooterHTML**.

Click on any of these topics to jump to them:

- Using TemplateNotInPattern
- Properties of the TemplateNotInPattern class
- Constructors for the TempateNotInPattern class


```
<desDD:DynamicPattern id="id" runat="server" PatternTemplateName="GridView" />
  <ItemsInPattern>
    <desDD:ScaffoldInPattern />
    <desDD:TemplateNotInPattern HTML="<br /><br />" />
    <desDD:ButtonsInPattern NamedPart="Buttons" />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

```
<desDD:DynamicPattern id="id" runat="server" PatternTemplateName="GridView" />
  <ItemsInPattern>
    <desDD:ScaffoldInPattern />
    <desDD:TemplateNotInPattern>
      <Template>
        <br />
        Bid: <asp:TextBox id="Bid" runat="server" />
      </Template>
    </desDD:TemplateNotInPattern>
    <desDD:ButtonsInPattern NamedPart="Buttons" />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

Adds a Select button as the first column of a DynamicListView, with a column header that has no text.

```
<desDD:DynamicListView id="id" runat="server" AutoGenerateButtons="None" />
  <ItemsInPatter>
    <desDD:TemplateNotInPattern>
        <Template>
          <td class="GridViewButtonColumn">
            <des:LinkButton ID="Select" runat="server" Text="Select"
            CommandName="Select" />
          </td>
        </Template>
        <HeaderTemplate>
          <td class="GridViewHeaderColumn"> </td>
        </HeaderTemplate>
    </desDD:TemplateNotInPattern>
    <desDD:ScaffoldInPattern />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

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

Using TemplateNotInPattern

To define HTML and web controls, use the **Template** property. Fill it with the desired markup. If you supply DynamicControls or DynamicLabels, explicitly assign their **DataField** properties.

```
<desDD:TemplateNotInPattern>
<Template>
   <br />
   <p>The answer is:</p>
   <desDD:DynamicControl id="Answer" runat="server" DataField="Answer" />
</Template>
</desDD:TemplateNotInPattern>
```

If you only have text or HTML to output, you can declare it as a string in the **HTML** property instead of defining a template.

```
<desDD:TemplateNotInPattern HTML="<br /><br />" />
```

If both the **HTML** and **Templates** properties are used, the **HTML** property value appears before the template.

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

Properties of the TemplateNotInPattern class

- **HTML** (string) – When assigned, this HTML is inserted into the page. If both **HTML** and **Template** are used, **HTML** will be output before **Template**. It defaults to "".
- **Template** (ITemplate) – The controls and HTML to output. It can contain DynamicControls and DynamicLabels. They should have their DataField property assigned.

  Use this syntax to define a template in ASP.NET Markup:

  ```
  <desDD:TemplateNotInPattern>
  <Template>
     HTML and web controls go here
  </Template>
  </desDD:TemplateNotInPattern>
  ```

  To assign it programmatically, you must create a class that implements `System.Web.UI.ITemplate`. Use the `ITemplate.InstantiateIn()` method to generate controls that belong in the Container. Create an instance of your class and either assign it to the **Template** property or through the TemplateNotInPattern constructor.
- **HeaderHTML** (string) – When assigned, this HTML is inserted into the page when the DynamicListView is generating a header row or when DynamicPattern has **PatternTemplateUsage** set to `Header`. If both **HeaderHTML** and **HeaderTemplate** are used, **HeaderHTML** will be output before the **HeaderTemplate**.

  If unassigned, it falls back to **HTML**.

  It defaults to "".
- **HeaderTemplate** (ITemplate) – The controls and HTML to output when the DynamicListView is generating a header row or when DynamicPattern has **PatternTemplateUsage** set to `Header`.

  If unassigned, it falls back to **Template**.
- **FooterHTML** (string) – When assigned, this HTML is inserted into the page when the DynamicListView is generating a Footer row or when DynamicPattern has **PatternTemplateUsage** set to `Footer`. If both **FooterHTML** and **FooterTemplate** are used, **FooterHTML** will be output before the **FooterTemplate**.

  If unassigned, it falls back to **HTML**.

  It defaults to "".
- **FooterTemplate** (ITemplate) – The controls and HTML to output when the DynamicListView is generating a Footer row or when DynamicPattern has **PatternTemplateUsage** set to `Footer`.

  If unassigned, it falls back to **Template**.

CONTINUED ON THE NEXT PAGE

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

- **ID** (string) – Optional identifier to make it easy to retrieve this object from a collection of ItemInPattern objects.

  It defaults to "".

  To get the object by its ID, call the `FindByID("ID")` method. Its declared on these classes: DynamicPattern, DynamicListView, DynamicFormView, and DataFieldListInPattern.

  Example

  Assign the **HTML** property on ID="NeedsHTML".

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
  PatternTemplateName="MyPatternTemplate" >
  <ItemsInPattern>
    <desDD:DataFieldInPattern DataField="columnname1" />
    <desDD:TemplateNotInPattern ID="NeedsHTML" />
  </ItemsInPattern>
</desDD:DynamicPattern>
```

  [C#]

```
BaseItemInPattern vItemInPattern = DynamicPattern1.FindByID("NeedsHTML");
if ((vItemInPattern != null) && (vItemInPattern is TemplateNotInPattern))
    ((TemplateNotInPattern)vItemInPattern).HTML = "<br /><br />";
```

  [VB]

```
Dim vItemInPattern As BaseItemInPattern = _
   DynamicPattern1.FindByID("NeedsHTML")
If (vItemInPattern IsNot Nothing) And _
   (TypeOf vItemInPattern Is TemplateNotInPattern) Then
   CType(vItemInPattern, TemplateNotInPattern).HTML = _
      "<br /><br />"
End If
```

- **Enabled** (Boolean) – Determines if the object is used or ignored. When `true`, it is used. When `false`, it and its children are not used. It defaults to `true`.

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

Constructors for the TempateNotInPattern class

These constructor declarations refer to the property names they set.

[C#]

```
TemplateNotInPattern()
TemplateNotInPattern(string pHTML)
TemplateNotInPattern(ITemplate pTemplate)
```

[VB]

```
New()
New(ByVal pHTML As String)
New(ByVal pTemplate As ITemplate)
```

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

The NamedStyle Class

The `PeterBlum.DES.DynamicData.NamedStyle` class is used to change various attributes in HTML tags within a Pattern Template. It provides a rule-based system to identify alternative styles. All NamedStyles have a name that matches to a **NamedStyle** property found in the Pattern Template, such as in a DynamicTabStyles control. You can develop rules based on data field name, category name, and user role name, that allow different appearances for the same named NamedStyle.

These objects are created in the **NamedStyles** collection of the DynamicPattern, DynamicListView, and DynamicFormView controls. Each has a name, which much match a named style declared in the Pattern Template.

As the Pattern Template generates its controls, it finds a matching NamedStyle object and merges its `class=`, `style=`, and other HTML attributes with its own, replacing any values defined within Pattern Template.

See "Named Styles: Your pages control the formatting".

Click on any of these topics to jump to them:

- Properties on the NamedStyle class
- Named Styles on DynamicPattern control
- Named Styles on DynamicListView control
- Named Styles on DynamicFormView control


```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
     PatternTemplateName="TwoColumns" DataField="ProductName" >
  <NamedStyles>
     <desDD:NamedStyle Name="LabelColumn" Style="width:150px;float:right;" />
     <desDD:NamedStyle Name="Row" CssClass="CustomRowStyle" />
  </NamedStyles>
</desDD:DynamicPattern>

<desDD:DynamicListView id="DynamicListView1" runat="server" >
  <NamedStyles>
     <desDD:NamedStyle Name="Row" DataField="Price" CssClass="PriceStyle" />
     <desDD:NamedStyle Name="Row" Category="Title" Style="font-weight:bold;" />
  </NamedStyles>
</desDD:DynamicListView>
```

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

Properties on the NamedStyle class

The NamedStyle class is subclassed from the `System.Web.UI.WebControls.Style` class. Many of the base class's properties are no longer shown. Instead, a style sheet class or style= property should define those properties, such as font, borders, and background.

For a guide on CSS styles, see https://developer.mozilla.org/en/CSS_Reference.

Click on any of these topics to jump to them:

- Identity Properties
- CSS Properties
- Other HTML Attribute Properties

Identity Properties

These properties let the Pattern Template locate your NamedStyle object.

- **Name** (string) – The name for the NamedStyle as defined by the Pattern Template. For example, "Row", "HeaderCell", or "FooterRow". Look at the Pattern Template to determine valid names and their purpose.

  You can specify more than one Name by using a pipe delimited list, like this: "`Row|HeaderRow|FooterRow`".

  Names use a case insensitive match.

  This property requires a value or the entire NamedStyle object will be ignored.

- **DataFields** (string) – Narrow the matching by the current data field that is being generated. This should contain the name of a DataField. It can contain multiple DataField names in a pipe delimited list.

  When assigned, data field matching is required.

  Use **ExcludeDataFields** to determine if a data field match means that data field must or must not be generated.

  Not all NamedStyles are used for data field generation, such as when creating a button. So only set it when the named style is used for data field generation. If both **DataFields** and **Categories** are assigned, only **DataFields** is used.

  DataFields use a case insensitive match.

  It defaults to "".

- **ExcludeDataFields** (string) – Determines what happens when the **DataFields** property matches to the data field being generated. Normally when the data field name is found, that NamedStyle is a match. This property can make that NamedStyle not match.

  When `true`, exclude the NamedStyle with any of these data fields names.

  When `false`, include the NamedStyle with any of these data fields names.

  It defaults to `false`.

- **Categories** (string) – Narrow the matching by the CategoryAttribute associated with the current data field that is being generated. This should contain the name of a category. It can contain multiple category names in a pipe delimited list.

  Data fields can have CategoryAttributes in their metadata to assist with setting up NamedStyles and Automatic Scaffolding. CategoryAttributes permit a multiple category names. Just one of those names must match just one name in the Categories property in order for the NamedStyle object to be included or excluded.

  When assigned, category matching is required.

  Use **ExcludeCategories** to determine if the category must or must not be found in the CategoryAttribute.

  Categories are only checked when the **DataField** is not assigned. Matching is case insensitive.

  It defaults to "".

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

- **ExcludeCategories** (Boolean) – Determines what happens when the **Categories** property contains a matching name to the CategoryAttribute's value. Normally when the Category name is found in the CategoryAttribute, that NamedStyle is a match. This property can make that NamedStyle not match.

  When `true`, exclude data fields with any of these category names.

  When `false`, include data fields with any of these category names.

  It defaults to `false`.

- **CategoriesWithNoMatchAreExcluded** (Boolean) – Determines what happens when the **Categories** property does *not* contain a matching name to the CategoryAttribute's value, including when the data field lacks a CategoryAttribute.

  When `true`, exclude data fields with any of these category names.

  When `false`, include data fields with any of these category names.

  It defaults to `true`.

- **Roles** (string) – Narrow the matching by the roles of the current user. This should contain the name of a role. It can contain multiple role names in a pipe delimited list.

  When assigned, role matching is required.

  Use **ExcludeRoles** to determine if a role match means that role must or must not be in the user's role list.

  Roles use a case insensitive match.

  It defaults to "".

- **ExcludeRoles** (string) – Determines what happens when the **Roles** property matches to the current user's roles. Normally when the role name is found, that NamedStyle is a match. This property can make that NamedStyle not match.

  When `true`, exclude the NamedStyle with any of these role names.

  When `false`, include the NamedStyle with any of these role names.

  It defaults to `false`.

- **Purpose** (string) – Descriptive information about the purpose of this field. This value was created to allow the Pattern Template to supply a description to the user within design mode. The user opens the NamedStyles editor when its empty and it prepopulates the list with items from the Pattern Template, along with this property describing each item. So generally use this as documentation.

- **Enabled** (Boolean) – Determines if the NamedStyle is enabled (active). It normally is. Set it to `false` as a convenient to remove items without deleting them, such as while you are testing various NamedStyles.

  It defaults to `true`.

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

CSS Properties

These properties populate the class= and `style=` attributes within the HTML tag.

For a guide on CSS styles, see https://developer.mozilla.org/en/CSS_Reference.

- **CssClass** (string) – The style sheet class name. It creates the class= attribute in the HTML tag. Generally use style sheets of the Style property for your CSS properties. If you assign values to the Style property, they will override the setting within the class defined here, unless that style is demarked "!IMPORTANT".

  #### Merging Style Sheet Classes

  CSS allows declaring multiple classes in the class= attribute, separated by spaces, like this: `class="class1 class2"`. Where both have a common style property, later class names override the earlier, unless the !IMPORTANT notation is used.

  When using Pattern Templates, this feature of the class= attribute is very useful. The Pattern Template usually declares its own style sheet class and your web form wants to modify it (instead of replace it). For example, you want to change the width style property but continue to use the existing class. You start by creating a new style sheet class with the modifications for the web form.

  While you can declare both classes in the **CssClass** property like this, `CssClass="PatternTemplateClass WebFormClass"`, use the plus (+) character and only the web form class name to merge to with whatever name the Pattern Template uses.

  `CssClass="+WebFormClass"` outputs `class="PatternTemplateClass WebFormClass"`

  `CssClass="WebFormClass+"` outputs `class="WebFormClass PatternTemplateClass"`

- **Style** (System.Web.UI.CssStyleCollection) – A list of cascading style sheet styles in the CSS syntax. They are assigned to the HTML tag within the `style=` attribute. This Style property will be merged with the DynamicStylesTag.Style property. When both have the same property name, the Named Style will be used.

  The CSS syntax is this pattern:

  ```
  propertyname:value; propertyname2:value2;
  ```

  For example:

  ```
  vertical-align:bottom; white-space:nowrap;
  ```

  The CSS property name and value must be identical to what the browser expects or they will not be used.

  You can locate the CSS properties here: https://developer.mozilla.org/en/CSS_Reference

  Warning: Any CSS properties defined in the style= attribute will override the setting within the class= attribute, unless that style sheet class is demarked "!IMPORTANT".

  When entering the **Style** property in ASP.NET markup, Intellisense is available to assist you in picking style names and defining their values.

  When working programmatically, the Style property can be assigned in two ways:

  To add a single string containing the CSS syntax that replaces the current value of Style, do this:

  ```
  DynamicStylesTag1.Style.Value = "propertyname:value; propertyname2:value2;"
  ```

  To add a single property, do this:

  ```
  DynamicStylesTag1.Style.Add("propertyname", "value")
  ```

CONTINUED ON THE NEXT PAGE

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

Smart Styles

When you define any of these style names, you may get the following behaviors:

text-align – When the HTML tag is `<td>` or `<th>`, this will be switched to an `align=` attribute if the browser does not support text-align on table cells.

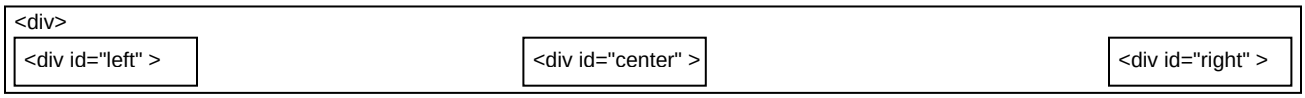
vertical-align – When the HTML tag is `<td>` or `<th>`, this will be switched to a `valign=` attribute if the browser does not support vertical-align on table cells.

Display – If following values are not supported by the browser, the display property is omitted: inline-block, table, table-row, table-cell, list-item, and inline-table.

Style Suggestions

- **Horizontal Alignment** – The `text-align` CSS attribute is recommended. However, it does not work when centering "block" type tags, like and <p> within another tag. Instead use `margin-left:auto;margin-right:auto;` around the inner block as described here: http://www.w3.org/Style/Examples/007/center.

  Suppose that you want to position three tags within another, where the first is left justified, the second is centered, and the third is right justified.

  Use the float style to establish left and right positioning on the div tags. Use the text-align style on the container div to center the rest. Recommendation: Position the HTML elements that use float before any that do not.

```
<div style="text-align" >
   <div id="left" style="float:left;" ></div>
   <div id="right" style="float:right;" ></div>
   <div id="center" ></div>
</div>
```

- **Vertical Alignment** – The `vertical-align` CSS attribute only works on table cells and elements with display:table-cell. (Further warning: Display:table-cell is not supported on many browsers.) Vertical alignment without tables can be tricky.

  Suggestion: When vertical alignment is needed, change to `<table>` and use the `vertical-align` style on the `<td>` tag.

  If tables are not suitable for you, here are a few articles to assist you:

  http://www.jakpsatweb.cz/css/css-vertical-center-solution.html

  http://phrogz.net/CSS/vertical-align/index.html

  Be aware that browsers differ in their support for style features like display: table-cell and display:inline-block. Even browsers that support table-cell may implement it differently. See http://www.quirksmode.org/css/display.html.

- **Preventing word wrap** – To keep text on the same line, use the `white-space:nowrap` style. It does not always work though.

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

Other HTML Attribute Properties

If you have other HTML tag attributes to assign, like `colspan=` or `valign=`, declare them as follows. The NamedStyle class does not define explicit property names for the multitude of possible HTML attributes, but it allows adding them in the ASP.NET markup or programmatically. (They cannot be defined in the Properties Editor.)

ASP.NET Markup

Define them as properties in the NamedStyles object. There will not be an intellisense or design mode support for these properties. If you are building XHTML pages, use lowercase for your property names.

```
<desDD:NamedStyle Name="DataColumn" CssClass="DataColumn"
    colspan="2" valign="bottom" >
```

Programmatically adding attributes

Use the `SetAttributes()` method on the NamedStyles object. It takes two parameters: style name and value. Both are strings. It will replace the same named property if already added.

```
DynamicStylesTag1.SetAttributes("colspan", "2")
```

```
DynamicStylesTag1.SetAttributes("valign", "bottom")
```

If you are building XHTML pages, use lowercase for your property names.

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

The DynamicListView Control

The DynamicListView control is the recommended DataBound control for list-style interfaces within DES Dynamic Data. While you can also use the GridView and ListView controls for these interfaces, both have limitations that are overcome by the DynamicListView control.

Quality	GridView	ListView	DynamicListView
Easy to setup	Very easy	Difficult	Very easy
Ability to customize HTML	Minimal	Complete	Complete
Supports automatic scaffolding	Yes	No	Yes

DynamicListView is really a blend of the ListView control with Pattern Templates. It subclasses from `System.Web.IU.WebControls.ListView`. As a result, it has every feature of ListView. Internally, it populates all of the Templates found on ListView with Pattern Templates. Pattern Templates offload all formatting into separate files (in the **[web application]\DynamicData\PatternTemplates** folder.) You provide a list of data fields to the DynamicListView (optionally using scaffolding) and it builds the HTML output by connecting each data field name to the controls defined in the Pattern Template.

If needed, you can switch to declaring controls directly in its Templates, making it return to its ListView roots.

The default DynamicListView control uses the **GridView.ascx** Pattern Template file which mimics the appearance of the GridView control. A second Pattern Template file, **ListOfDetailViews.ascx**, shows each record in the style of the DetailsView control.

Click on any of these topics to jump to them:

- Using the DynamicListView Class
- Adding the DynamicListView Class
- Properties of the DynamicListView Class


```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
   DataSourceID="ListDataSource" >
</desDD:DynamicListView>

<desDD:DynamicListView ID="DynamicListView1" runat="server"
   DataSourceID="ListDataSource" DataKeyNames="ProductID" >
   <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="ProductName"  />
      <desDD:DataFieldInPattern DataField="UnitPrice"  />
      <desDD:DataFieldInPattern DataField="QuantityPerUnit"  />
      <desDD:DataFieldInPattern DataField="ReorderLevel"  />
      <desDD:DataFieldInPattern DataField="UnitsInStock"  />
      <desDD:DataFieldInPattern DataField="Discontinued"  />
   </ItemsInPattern>
   <NamedStyles>
      <desDD:NamedStyle DataField="QuantityPerUnit" Style="white-space:nowrap;" />
   </NamedStyles>
</desDD:DynamicListView>
```

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

Using the DynamicListView Class

Click on any of these topics to jump to them:

- Getting Started
- Comparing to ListView
- How to use the PartsPatternTemplate with a DynamicListView
- Parts Overview for PartsPatternTemplate
- Establishing a DataSource
- Defining a list of data fields
- Changing the appearance
- Offering Paging
- Grouping records to provide a tiled interface

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

Getting Started

Like all DataBound controls, your web form must contain a Dynamic Data compatible Data Source and a DynamicDataManager control with a DataBound Control Adapter that identifies your DynamicListView control. If the list supports editing, also add a ValidationSummary control.

Here is the basic markup.

```
<desDD:DynamicDataManager ID="DynamicDataManager1" runat="server" >
  <Adapters>
    <desDD:DynamicListViewAdapter DataBoundControlID="DynamicListView1" />
  </Adapters>
</desDD:DynamicDataManager>

<desDD:DynamicListView ID="DynamicListView1" runat="server"
  DataSourceID="ListDataSource" >
</desDD:DynamicListView>

<asp:LinqDataSource ID="ListDataSource" runat="server"
  ContextTypeName="DataContext" TableName="tablename" >
</asp:LinqDataSource>
```

Note: When the PatternTemplateName property is not specified on the DynamicListView control, it defaults to the GridView.ascx PatternTemplate file.

This example adds editing capabilities. Both the DataSource and Adapter have new properties, plus the ValidationSummary has been added.

```
<desDD:DynamicDataManager ID="DynamicDataManager1" runat="server" >
  <Adapters>
    <desDD:DynamicListViewAdapter DataBoundControlID="DynamicListView1"
      SupportsEditMode="true" ValidationSummaryControlID="ValidationSummary1" />
  </Adapters>
</desDD:DynamicDataManager>

<des:ValidationSummary ID="ValidationSummary1" runat="server" InAJAXUpdate="True"/>

<desDD:DynamicListView ID="DynamicListView1" runat="server"
  DataSourceID="ListDataSource" >
</desDD:DynamicListView>

<asp:LinqDataSource ID="ListDataSource" runat="server"
  ContextTypeName="DataContext" TableName="tablename" EnableUpdate="true" >
</asp:LinqDataSource>
```

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

Comparing to ListView

As a direct subclass of the ListView control, DynamicListView supports most of the capabilities describing ListView. Here are some sources of information:

- Control overview
- Class overview

There are exceptions because this control is designed to work within Dynamic Data. Here are some issues to be aware of:

- The Pattern Template file must inherit from `PeterBlum.DES.DynamicData.PartsPatternTemplate` as this Pattern Template supplies all of the parts that make up a list of records. It even supports these ListView specific Templates: GroupTemplate, EmptyItemTemplate, EmptyDataTemplate, and GroupSeparatorTemplate.
- The DataSource must support Dynamic Data, such as LinqDataSource, EntityDataSource, and DynamicObjectDataSource.
- Most template properties are populated automatically, following the rules of the **ItemsInPattern** and **NamedStyles** collections. If you use the template properties directly, your template code will be used instead of the automatic code.

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

How to use the PartsPatternTemplate with a DynamicListView

DynamicListView demands that you select a Pattern Template file which inherits from `PeterBlum.DES.DynamicData.PartsPatternTemplate`. The PartsPatternTemplate supplies all of the parts that make up a list of records. It even supports these ListView specific Templates: GroupTemplate, EmptyItemTemplate, EmptyDataTemplate, and GroupSeparatorTemplate.

Explore the **GridView.ascx** and **ListOfDetailViews.ascx** Pattern Template files in the **[web application]/DynamicData/PatternTemplates** folder to see how each define their output. The overall structure of these files looks like this:

```
<%@ Control Language="C#" Inherits="PeterBlum.DES.DynamicData.PartsPatternTemplate"
      Usage="PartsForList" %>
<%@ Register tagprefix="desDD" assembly="PeterBlum.DES.DynamicData"
      namespace="PeterBlum.DES.DynamicData" %>

<script runat="server">
   public override DynamicPartsPatterns GetDynamicPartsPatterns()
   {
      return DynamicPartsPatterns1;
   }
</script>

<desDD:DynamicPartsPatterns ID="DynamicPartsPatterns1" runat="server">

      <Various Parts and optional NamedParts>

</desDD:DynamicPartsPatterns>
```

You generally edit the inner content of the DynamicPartsPatterns control by adding or modifying Parts tags. Here are a few examples taken from **GridView.ascx**.

```
<DataCell NamedStyle="DataCell" CssClass="<%#cCssDataCell %>"
      Style="vertical-align:top;" >

   <Template>
      <desDD:DynamicControl ID="DynamicControl1" runat="server" />
   </Template>

   <NamedParts>
      <desDD:DataCellNamedPatternPart Name="TwoStacked">
         <Template>
            <desDD:DynamicControl ID="DynamicControl1" runat="server" /><br />
            <desDD:DynamicControl ID="DynamicControl2" runat="server" />
         </Template>
      </desDD:DataCellNamedPatternPart>
      <desDD:DataCellNamedPatternPart Name="ShowLabel">
         <Template>
            <desDD:DynamicLabel ID="DynamicLabel1" runat="server" />
         </Template>
      </desDD:DataCellNamedPatternPart>
   </NamedParts>

</DataCell>

<NoDataToShow Tag="Div" NamedStyle="NoDataToShow"
      CssClass="<%# cCssNoDataToShow %>" Message="No records were found." >
</NoDataToShow>
```

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

Parts Overview for PartsPatternTemplate when used with Lists

Here is a quick overview of the Parts where you define the HTML output.

Part tag	Description	In Template
`<Container>`	The HTML tag around the entire control. Specify its **Tag** property to define the HTML tag. It does not supply a Template property for any of its inner content because the rest of the tags are the inner content.	LayoutTemplate or GroupTemplate
`<DataRow>`	Describes the content for a single record. This Part tag provides three templates that define different output based on the data entry mode. You can declare the DynamicControls here, but if you want them to be generated for each data field automatically, use the <DataCell> for that and use <DataRow> to define the HTML tag that encloses the list of <DataCell> parts.	All ItemTemplates, including Edit, Insert, Alternating, and Selected
`<DataCell>`	Describes the content of a single field in the record. This Part tag provides three templates that define different output based on the data entry mode. It should have at least one DynamicControl. It is used when there is no Template defined in <DataRow> or if the Template includes a PlaceHolder for the <DataCell> parts.	Depends on <DataRow>
`<Buttons>`	Describes the element that shows buttons for the record. The buttons depend on the data entry mode, such as Edit and Details Buttons for ReadOnly mode and Save and Cancel Buttons for Insert and Edit modes. This part often contains a DynamicButtons control which knows the correct buttons to show based on the mode and various other settings in your form. This part is only used when the <DataRow> also allows the output of <DataCells>. It depends on either the **DynamicListView.AutoGenerateButtons** property or the presence of a ButtonsInPattern object in the **ItemsInPattern** collection.	Depends on <DataRow>
`<HeaderRow>`	Describes the content for a header prior to the list of records. It often includes column titles. You can declare the DynamicColumnTitle controls here, but if you want them to be generated for each data field automatically, use the <HeaderCell> for that and use <HeaderRow> to define the HTML tag that encloses the list of <HeaderCell> parts.	LayoutTemplate or GroupTemplate
`<HeaderCell>`	Describes the content of a single field's header, such as a column title. It usually has a DynamicColumnTitle or DynamicLabel which will automatically be assigned the data field name. It is used when there is no Template defined in <HeaderRow> or if the Template includes a PlaceHolder for the <HeaderCell> parts.	Depends on <HeaderRow>
`<HeaderButtons>`	Describes a header for a column containing <Buttons> parts. It is often the column title for buttons. If it does not need a title, it often uses `" "`.	Depends on <HeaderRow>

CONTINUED ON THE NEXT PAGE

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

Part tag	Description	In Template
`<FooterRow>`	Describes the content for a Footer after to the list of records. If you want one element generated for each data field, use the <FooterCell> part.	LayoutTemplate or GroupTemplate
`<FooterCell>`	Describes the content of a single field's Footer. It is used when there is no Template defined in <FooterRow> or if the Template includes a PlaceHolder for the <FooterCell> parts.	Depends on <FooterRow>
`<FooterButtons>`	Describes a Footer for a column containing <Buttons> parts.	Depends on <FooterRow>
`<DataRowSeparator>`	A row that separates each record. If you want one element generated for each data field, use the <DataRowSeparatorCell> part.	ItemSeparator-Template
`<DataRow-SeparatorCell>`	A single field's content for the <DataRowSeparator>.	Depends on <DataRowSeparator>
`<NoDataToShow>`	Used when there is no data found. It is often a message like "No data found."	EmptyDataTemplate
`<GroupContainer>`	When **EnableGrouping** is `true`, the ListView's GroupTemplate feature is setup. <GroupContainer> is the HTML tag that encloses a group. Specify its **Tag** property to define the HTML tag. It does not supply a Template property for any of its inner content because the rest of the tags are the inner content.	LayoutTemplate
`<GroupHeader>`	The header for the grouping feature. HTML that separates the Group's opening tag in <GroupContainer> from the first Item of actual data. It is often used for titles. It does not support DynamicControls.	LayoutTemplate
`<GroupFooter>`	The footer for the grouping feature. HTML that separates the last Item of actual data from the Group's closing tag in <GroupContainer>.	LayoutTemplate
`<GroupSeparator>`	HTML that separates groups from each other. Associated with the ListView GroupSeparatorTemplate feature.	GroupSeparator-Template
`<EmptyItem>`	Defines the appearance where an item would normally go in a group, but there is no data for that item. It can contain the entire HTML or let <EmptyItemCells> define the appearance for each cell associated with a DataFieldInPattern object.	EmptyItemTemplate
`<EmptyItemCell>`	This a cell within a row that describes an record without any data. It only appears when <EmptyItem> indicates its use. One is generated for each DataFieldInPattern object.	Depends on <EmptyItem>

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

Establishing a DataSource

Always set the **DataSourceID** property to the ID of a DataSource control that is compatible with Dynamic Data. The DataSource control is expected to be in the same naming container as the DynamicListView control.

Establishing Edit mode

If DynamicListView should offer editing capabilities, set the **EnableUpdate** property to `true` in the DataSource and **SupportsEditMode** property to `true` on the DynamicListViewAdapter object.

Establishing Insert mode

If DynamicListView should offer inserting capabilities, set the **EnableInsert** property to `true` in the DataSource and **SupportsInsertMode** property to `true` on the DynamicListViewAdapter object. You will also need to add a button that starts Insert mode. See below for details on setting up buttons.

You can have a row always in Insert mode when there are no rows in edit mode by using the **AutoInsertItemMode** property. When used, it controls the ListView's **InsertItemPosition** property (which when used adds a row either before or after the data rows for insert). It has these values:

- `No` - User must always use **InsertItemPosition** to show and hide this row.
- `First` - The row is shown first in the list, unless another row is in Edit mode.
- `Last` - The row is show last in the list, unless another row is in Edit mode.
- `AutoDeactivate` - The user must set **InsertItemPosition** or call `DynamicDataManager.ChangeMode(Insert)` to activite Insert mode, but the grid does the work to remove the row when editing is finished. Many actions finish editing: sorting, starting an edit row, selecting a row, etc. This is the default.

Here are ways to establish the Insert button.

- Use the DynamicControlPanel – Add the NewCommandButtonField or DynamicButtonsField to the DynamicControlPanel. It will show the New Record button and set it up to work with the DynamicListView.

```
<desDD:DynamicControlPanel ID="BottomControlPanel" runat="server"
   PagedControlID="DynamicListView1" PageSize="5" ContainerTag="Div" >
   <Fields>
      <desDD:DynamicButtonsField />
   </Fields>
</desDD:DynamicControlPanel>
```

- Use a DynamicHyperLink – When using Dynamic Data Routing with a Hyperlink, choose the DynamicHyperlink control. *Note: Requires ASP.NET 4 or a prerelease:*

```
<asp:DynamicHyperLink runat="server" Action="Insert" Text="New" />
```

- Use a DES Button, LinkButton, or ImageButton – Attach the button's Click event handler to call this code:

  [C#]

```
protected void InsertButton_Click(object sender, EventArgs e)
{
  DynamicDataManager1.ChangeMode(DynamicListView1, DataBoundControlMode.Insert);
}
```

  [VB]

```
Protected Sub InsertButton_Click(ByVal sender As Object, ByVal e As EventArgs)
  DynamicDataManager1.ChangeMode(DynamicListView1, DataBoundControlMode.Insert)
End Sub
```

  Replace DynamicDataManager1 and DynamicListView1 with references to the actual controls.

  The `ChangeMode()` method will evaluate the DataBound Control Adapter's rules for using insert mode. It may go to another page using Dynamic Data Routing, a URL that you have specified, or use the same DynamicListView control to provide a row in Insert mode.

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

Establishing Row Deletion

If DynamicListView should offer a Delete button, set the **EnableDelete** property to `true` in the DataSource.

Establishing Row Selection

The ListView supports the concept of a selected row, making it work like a listbox. The row has these qualities:

- It's **SelectedIndex** property identifies a row with a selection. -1 indicates no selection.
- The selected row has a different style.
- Rows offer the Select button to mark the row as selected. (Only one row can be selected at a time.)

To support row selection, set the **AllowSelectedRows** property to `true`.

While there should be a Select button on each row, the other row-specific commands can be moved to the DynamicControlPanel and operate based on the selected row. Row specific commands include Details, Edit, Delete, Save and Cancel. The **[DES product folder]/DynamicData/language/Samples/DynamicListViewWithSelect.aspx** webform demonstrates this.

To set it up, use these elements:

- Set **AllowSelectedRows** to true on DynamicListView.
- Set **ButtonsNamedPart** property to a NamedPart in the `<Buttons>` section that only supplies a Select button. Both **GridView.ascx** and **ListOfDetailViews.ascx** define these names: "SelectButton" and "SelectLinkButton".
- Add the DynamicButtonsField to the DynamicControlPanel.

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
  DataSourceID="ListDataSource" DataKeyNames="ProductID"
  AllowSelectingRows="true" ButtonsNamedPart="SelectLinkButton" >
</desDD:DynamicListView>

<desDD:DynamicControlPanel ID="ControlPanel" runat="server"
  PagedControlID="DynamicListView1" PageSize="10">
  <Fields>
     <desDD:DynamicButtonsField />
  </Fields>
</desDD:DynamicControlPanel>
```

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

Defining a list of data fields

The DynamicListView displays data from a data source. The data source provides it with the data – the table and rows requested by a query. The DynamicListView needs you to determine which columns (data fields) are displayed and the order. There are two ways to approach this:

- Use Automatic Scaffolding. If you leave the **ItemsInPattern** collection empty, scaffolding is used unless you set **AutoScaffoldDataFields** to `False`. If you have items in **ItemsInPattern**, set the **AutoScaffoldDataFields** property to `True`. See "Automatic Scaffolding Overview".

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
  DataSourceID="ListDataSource" >
</desDD:DynamicListView>
```

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
  DataSourceID="ListDataSource" AutoScaffoldDataFields="True" >
  <ItemsInPattern>
    <desDD:NoDataInPattern NamedPart="Logo"  />
  </ItemsInPattern>
</desDD:DynamicListView>
```

- Populate the **ItemsInPattern** collection with DataFieldInPattern and other ItemInPattern objects.

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
  DataSourceID="ListDataSource" >
  <ItemsInPattern>
    <desDD:DataFieldInPattern DataField="ProductName"  />
    <desDD:DataFieldInPattern DataField="UnitPrice"  />
    <desDD:DataFieldInPattern DataField="QuantityPerUnit"  />
    <desDD:DataFieldInPattern DataField="ReorderLevel"  />
    <desDD:DataFieldInPattern DataField="UnitsInStock"  />
    <desDD:DataFieldInPattern DataField="Discontinued"  />
  </ItemsInPattern>
</desDD:DynamicListView>
```

See the following topics:

- Defining ItemInPattern objects in the ItemsInPattern collection
- Defining the Data Field names shown by the PatternTemplate
- Adding HTML and other web controls between data fields
- Passing properties to the Field Templates

The ItemInPattern classes are:

- The DataFieldInPattern Class
- The DataFieldListInPattern Class
- The ScaffoldInPattern Class
- The ButtonsInPattern Class
- The LabelInPattern Class
- The NoDataFieldsInPattern Class
- The TemplateNotInPattern Class

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

Changing the appearance

The HTML output comes from the Pattern Template file. It defines default style sheet classes and HTML formatting rules for the HTML and web controls that it contains. Pattern Templates can be customized by editing them. However, that may change a Pattern Template that is correctly formatted for another webform's usage. Instead consider these approaches.

Click on any of these topics to jump to them:

- Choosing the PatternTemplate
- Using Named Styles to adjust the formatting
- Buttons
- Header Row and Column Titles
- Footer Row
- Using ListView's Templates instead of automatic generation

Choosing the PatternTemplate

The DynamicListView works with any Pattern Template that subclasses from `PeterBlum.DES.DynamicData.PartsPatternTemplate`. It ships with two Pattern Template files designed with a list of records in mind: **GridView.ascx** (the default) and **ListOfDetailViews.ascx**. If you want to change from the default, set its name in the **PatternTemplateName** property.

- Using GridView.ascx PatternTemplate
- Using ListOfDetailViews.ascx PatternTemplate
- Customizing and creating your own PatternTemplates

If you want to change an existing Pattern Template file, consider making a new file because you may break web forms already using the Pattern Template file. Duplicate the Pattern Template file and edit the duplicate. Assign the file name of the new file in the **PatternTemplateName** property. The file extension ".ascx" is optional.

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
  DataSourceID="ListDataSource" RowsPatternTemplateName="GridViewStyle2" >
</desDD:DynamicListView>
```

If you created the file in a different folder than **[web application]/DynamicData/PatternTemplates**, use the **PatternTemplatesFolderUrl** property to specify the path to that folder.

The DynamicListView control offers more parts than you may want. It offers these properties to use or omit the parts:

- **ShowHeaderRow** – Shows the `<HeaderRow>` Part. It is shown by default.
- **ShowFooterRow** – Shows the `<FooterRow>` Part. It is hidden by default.
- **AlternatingRowStyles** – Determines if every other row uses an alternative style sheet to establish its appearance. When true, the AlternatingItemTemplate is used.
- **AutoGenerateButtons** – Determines if the `<Buttons>` part is added either before or after the data fields. By default, it is set to `First` to generate items before.
- **ShowDataRowSeparator** – Determines if the `<DataRowSeparator>` part is inserted between each record's data.
- **EnableGrouping** – Enables the ListView's Grouping feature where the **LayoutTemplate** describes a container around a list of data rows and their header and footer. **GroupTemplate** contains the actual appearance for output of data rows, taking the same formatting originally intended for the **LayoutTemplate**. See http://msdn.microsoft.com/en-us/library/bb398790.aspx.

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

Using Named Styles to adjust the formatting

Named Styles let you override the style sheet and other formatting rules on individual elements within the Pattern Template without modifying the Pattern Template. Instead, you declare a list of NamedStyle objects in the **NamedStyles** collection of the DynamicListView control.

Evaluate the Pattern Template file you have selected to see if you can use Named Styles. The Pattern Template files shipped with DES Dynamic Data all supply numerous Named Styles.

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
   DataSourceID="ListDataSource" >
   <NamedStyles>
      <desDD:NamedStyle Name="LabelControl" CssClass="SmallLabel" />
      <desDD:NamedStyle Name="Row" Category="Title" CssClass="Title" />
      <desDD:NamedStyle Name="Row" DataField="QuantityPerUnit"
           Style="white-space:nowrap;" />
   </NamedStyles>
</desDD:DynamicListView>
```

Because a NamedStyle can evaluate data field, category, and user role names, you may specify several NamedStyles with the same name, but different combinations of the other evaluators. It will always use the most specific match over a less specific match. For example, in the above example, it uses the third NamedStyle when “DataColumn” is being generated. If that is not generating output for “QuantityPerUnit”, it then uses the second NamedStyle. If two NamedStyles could match and have the same number of evaluators, it uses the order specified in the **NamedStyles** property.

See the following topics:

- Named Styles: Your pages control the formatting
- NamedStyles on the GridView PatternTemplate
- NamedStyles on the ListOfDetailViews PatternTemplate
- The DynamicStylesTag Control
- The NamedStyle Class

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

Buttons

By default, the **GridView.ascx** and **ListOfDetailViews.ascx** Pattern Templates use LinkButtons for buttons. If you want to change to standard Buttons, set the **ButtonsNamedPart** property to "`Buttons`".

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
   DataSourceID="ListDataSource" ButtonsNamedPart="Buttons" >
</desDD:DynamicListView>
```

Buttons appear in the first column by default. To position the buttons in a different column, use the **AutoGenerateButtons** property. It defaults to `First`. Use `Last` or `FirstLast` to offer buttons in those columns. Use `None` if you will add a ButtonsInPattern object whose **NamedPart** property specifies your interface for buttons within the **ItemsInPattern** collection. This allows you to position them anywhere.

If the predefined appearances for buttons don't match your requirements, you can create another interface in the `<Buttons>` Part of the PartsPatternTemplate.

Buttons can get their formatting through their own Pattern Template files. DES Dynamic Data includes these files for Button and LinkButton output: **ButtonsForList.ascx** and **LinkButtonsForList.ascx**. The **GridView.ascx** and **ListOfDetailViews.ascx** Pattern Templates declare two NamedParts that map to these files. They are "`Buttons`" and "`LinkButtons`".

```
<Buttons PatternTemplateName="LinkButtonsForList" other properties >
   <NamedParts>
      <desDD:ButtonsNamedPatternPart Name="LinkButtons"
         PatternTemplateName="LinkButtonsForList" />
      <desDD:ButtonsNamedPatternPart Name="Buttons"
         PatternTemplateName="ButtonsForList" />
   </NamedParts>
</Buttons>
```

To use different formatting with Pattern Templates.

1. Duplicate one of the Button PatternTemplate files and edit it.
2. Add a NamedPart into the `<Buttons>` Part of the PartsPatternTemplate:

```
<Buttons PatternTemplateName="LinkButtonsForList" other properties >
   <NamedParts>
      <desDD:ButtonsNamedPatternPart Name="LinkButtons"
         PatternTemplateName="LinkButtonsForList" />
      <desDD:ButtonsNamedPatternPart Name="Buttons"
         PatternTemplateName="ButtonsForList" />
      <desDD:ButtonsNamedPatternPart Name="newname"
         PatternTemplateName="newPatternTemplate" />
   </NamedParts>
</Buttons>
```

You can also add a DynamicButtons control to the Template property of the `<Buttons>` Part or one of its NamedParts.

```
<Buttons PatternTemplateName="LinkButtonsForList" other properties >
   <NamedParts>
      <desDD:ButtonsNamedPatternPart Name="LinkButtons"
         PatternTemplateName="LinkButtonsForList" />
      <desDD:ButtonsNamedPatternPart Name="Buttons"
         PatternTemplateName="ButtonsForList" />
      <desDD:ButtonsNamedPatternPart Name="newname">
         <Template>
            <desDD:DynamicButtons id="DynamicButtons1" runat="server" properties />
         </Template>
      </desDD:ButtonsNamedPatternPart>
   </NamedParts>
</Buttons>
```

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

Header Row and Column Titles

Note: The PartsPatternTemplate file may permanently disable the Header by setting the Enabled property to false within its <HeaderRow> Part. The ListOfDetailViews.ascx Pattern Template is setup this way.

The header row is generally used to show column titles. It is optional and shown by default. If you want to hide it, set **ShowHeaderRow** to `false`. The column titles come from the list of DataFieldInPattern objects in the **ItemsInPattern** collection. If you want to change the title from its default, use the **HeaderText** property of DataFieldInPattern.

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
   DataSourceID="ListDataSource" >
   <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="ProductName"  />
      <desDD:DataFieldInPattern DataField="UnitPrice" HeaderText="Price"  />
      <desDD:DataFieldInPattern DataField="QuantityPerUnit"  />
      <desDD:DataFieldInPattern DataField="ReorderLevel"  />
      <desDD:DataFieldInPattern DataField="UnitsInStock"  />
      <desDD:DataFieldInPattern DataField="Discontinued"  />
   </ItemsInPattern>
</desDD:DynamicListView>
```

By default, the **GridView.ascx** Pattern Template uses a DynamicColumnTitles control to show and sort by data field name. Sorting is determined by the data type of the column or if the SortableColumnAttribute is applied. If you want to change to a label that you cannot sort, set the **HeaderCellNamedPart** property to "`NoSort`". You can create other NamedParts in the `<HeaderCell>` Part which can be specified in the **HeaderCellNamedPart** property.

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
   DataSourceID="ListDataSource" HeaderCellNamedPart="NoSort" >
</desDD:DynamicListView>
```

The column title over the buttons column is determined by the `<HeaderButtons>` Part. If the Part includes a NamedPart that you want to use, set its name in the **HeaderButtonsNamedPart** property.

You can create your own header formatting within the **HeaderTemplate** property of the DynamicListView. In fact, you can mix the automatically generated headers with your own formatting as a way to enhance the automatically generated headers by using the **HeaderRules** property. This enumerated type has these values:

- `Normal` - If **HeaderTemplate** is defined, use it. Otherwise use the automatic generated header row. This is the default.
- `AutoOnly` - Only use the automatic generated header row.
- `TemplateOnly` - Only use the **HeaderTemplate**. If not defined, use nothing.
- `AutoThenTemplate` - First output the automatic generated header row. Then output the **HeaderTemplate**.
- `TemplateThenAuto` - First output the **HeaderTemplate**. Then output the automatic generated header row.

This example uses the automatic row followed by a line using an `<hr />` tag that goes across all columns.

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
   DataSourceID="ListDataSource" HeaderRules="AutoThenTemplate">
   <HeaderTemplate>
   <tr>
      <td colspan="999">
         <hr />
      </td>
   </tr>
   </HeaderTemplate>
</desDD:DynamicListView>
```

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

Footer Row

Note: The PartsPatternTemplate file may permanently disable the Footer by setting the Enabled property to false within its <FooterRow> Part. The ListOfDetailViews.ascx Pattern Template is setup this way.

The Footer row is optional and hidden by default. If you want to show it, set **ShowFooterRow** to `true`.

By default, the **GridView.ascx** Pattern Template uses a DynamicLabel controls without any text assigned. You are expected to use the FooterText property on either DataFieldInPattern or LabelInPattern objects to supply the desired text.

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
   DataSourceID="ListDataSource" >
   <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="Columnname1"
         FooterText="<hr />" />
      <desDD:DataFieldInPattern DataField="Columnname2"
         FooterText="<hr />" />
      <desDD:DataFieldInPattern DataField="Columnname3"
         FooterText="<hr />" />
   </ItemsInPattern>
</desDD:DynamicListView>
```

You can create alternative output by using NamedParts in the `<FooterCell>` Part. Specified its name in the **FooterCellNamedPart** property on DynamicListView.

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
   DataSourceID="ListDataSource" FooterCellNamedPart="Calculation" >
</desDD:DynamicListView>
```

The text under the buttons column is determined by the `<FooterButtons>` Part. If the Part includes a NamedPart that you want to use, set its name in the **FooterButtonsNamedPart** property.

You can create your own Footer formatting within the **FooterTemplate** property of the DynamicListView. In fact, you can mix the automatically generated Footers with your own formatting as a way to enhance the automatically generated Footers by using the **FooterRules** property. This enumerated type has these values:

- `Normal` - If **FooterTemplate** is defined, use it. Otherwise use the automatic generated Footer row. This is the default.
- `AutoOnly` - Only use the automatic generated Footer row.
- `TemplateOnly` - Only use the **FooterTemplate**. If not defined, use nothing.
- `AutoThenTemplate` - First output the automatic generated Footer row. Then output the **FooterTemplate**.
- `TemplateThenAuto` - First output the **FooterTemplate**. Then output the automatic generated Footer row.

The list can have a footer row by declaring the desired HTML and web controls within the **FooterTemplate** property.

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

Using ListView's Templates instead of automatic generation

DynamicListView has numerous Template properties to generate its parts. When they are not assigned, DynamicListView generates their content for you, by using a DynamicPattern control and the PartsPatternTemplate file. If you want to customize the Template directly, assign the desired HTML and web controls to it. Most use this to generate content from the PartsPatternTemplate:

```
<TemplateName>
  <desDD:DynamicPattern id="id" runat="server"
    PatternTemplateName="patterntemplatename" PartsToGenerate="PartsToGenerate" >
  </desDD:DynamicPattern>
</TemplateName>
```

It's easy to start your own markup with the markup that these controls would generate automatically.

1. Open the web form in Visual Studio and switch to Design mode.
2. Click on the SmartTag button.
3. Choose the appropriate the **Manage Auto Generation of Templates** command.
4. From the list, change templates from "Auto Generate" to "From Template" or back.

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

Passing properties to be used by the PatternTemplate's own code

Pattern Template can expose public properties that let you customize their behavior. For example, the **DetailsView.ascx** PatternTemplate has the Boolean property **HeaderLabelColumnHTML** which specifies the text of the label in the HeaderRow above the Label column. *Note: You won't be using DetailsView with DynamicListView. Its here as an example.*

[C#]

```
public string HeaderLabelColumnHTML
{
   get { return Host.GetAttributeAsString("HeaderLabelColumnHTML",
      cDefaultHeaderLabelColumnHTML); }
}
private const string cDefaultHeaderLabelColumnHTML = "Name";
```

[VB]

```
Public ReadOnly Property HeaderLabelColumnHTML() As String
   Get
      Return Host.GetAttributeAsString("HeaderLabelColumnHTML",
         cDefaultHeaderLabelColumnHTML)
   End Get
End Property
Private Const cDefaultHeaderLabelColumnHTML As String = "Name"
```

The value of this property is assigned on the DynamicListView control and retrieved by your property like this:

```
<desDD:DynamicListView id="DynamicListView1" runat="server"
   PatternTemplateName="DetailsView" HeaderLabelColumnHTML="Field Name" />
```

Just declare the property name and value within the ASP.NET markup of the DynamicListView control. Some of the properties it already knows about and lists in the Properties Editor of Visual Studio. Others – especially those you created – are not known but still can be specified.

When programmatically assigning this property, you must pass it through the `Add()` method of the **Attributes** property:

```
DynamicListView1.Attributes.Add("HeaderLabelColumnHTML", "Field Name")
```

Passing Properties specific to an ItemInPattern object

When using a PartsPatternTemplate, these properties can be passed along on individual DataFieldInPattern, DataFieldListInPattern, ButtonsInPattern, NoDataFieldsInPattern, and TemplateNotInPattern objects to be used by code within the Pattern Template in their **OnPartCreating** or **OnPartCreated** event handler methods.

The Pattern Template must call the `GetAttributeAsNative()` or `GetAttributeAsString()` methods on the *Args* parameter object passed to the **OnPartCreating** or **OnPartCreated** event handlers. Good documentation on the property names it supports is vital.

When the page developer identifies such a property, they specify the property name and value within the ASP.NET markup of the *xyz*InPattern object. In this example, the "ColumnTitle" NamedPart supports a property named "Title" that takes a string.

```
<desDD:DynamicListView id="DynamicListView1" runat="server"
   PatternTemplateName="MyPatternTemplate" >
   <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="columnname1" UIHint="FancyText"
         NamedPart="ColumnTitle" Title="my title" />
   </ItemsInPattern>
</desDD:DynamicListView>
```

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

Offering Paging

Paging is often a necessity for lists. Use the DynamicControlPanel control to create a wide variety of paging interfaces. Place the control above, below, or include the DataBound control as desired. When inside the DataBound control, the DynamicControlPanel can be inside the **HeaderTemplate**, the **FooterTemplate**, the `<HeaderRow>` Part or the `<FooterRow>` Part. You can have several on the page. Those that are outside of the DynamicListView must have their **DataBoundControlID** property assigned to the ID of the DynamicListView.

Here is the DynamicListView with DynamicControlPanel from the **List.aspx** Page Template:

```
<desDD:DynamicListView ID="RecordsList" runat="server"
   DataSourceID="ListDataSource" >
</desDD:DynamicListView>
<desDD:DynamicControlPanel ID="DynamicControlPanel1" runat="server" PageSize="10"
   PagedControlID="RecordsList" >
   <Fields>
      <desDD:PreviousPagerField />
      <desDD:InfoPagerField />
      <desDD:NextPagerField />
   </Fields>
</desDD:DynamicControlPanel>
```

See “The DynamicControlPanel Control”.

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

Grouping records to provide a tiled interface

The ListView control offers its **GroupTemplate** property to assist in developing a more complex user interface as described here:

ListView.GroupTemplate Property on MSDN

Using ASP.NET 3.5's ListView and DataPager Controls: Grouping Data

Using the ListView control in Tiled mode (Part 1)

Grouping is supported with DynamicListView. Set its EnableGrouping property to true and use a PartsPatternTemplate that has been setup to support Grouping by defining formatting in these Parts:

`<GroupContainer>` – Forms the HTML tag that encloses a group.

`<GroupHeader>` – HTML between the opening tag and first item.

`<GroupFooter>` – HTML between the last item and closing tag.

`<GroupSeparator>` – HTML separating each group.

`<EmptyItem>` and `<EmptyItemCell>` – HTML representing an item when there is no data to show for that data.

Use the **GroupItemCount** property to determine how many items appear per group.

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

Setting up a PartsPatternTemplate to tile left-to-right first

This interface will look like:

Record 1's contents	Record 2's contents	Record 3's contents
Record 4's contents	Record 5's contents	Record 6's contents

1. Start with the **ListOfDetailViews.ascx** Pattern Template or something similar. (This user interface is not effective with a single row per data record like that offered with **GridView.ascx**.)
2. Here are the Parts definitions that are required. They are already setup in **ListOfDetailsView.ascx**.

```
<GroupContainer Tag="Table" />

<GroupHeader Tag="None">
<Template>
  <tr>
    <td>
</Template>
</GroupHeader>

<GroupFooter Tag="None">
<Template>
    </td>
  </tr>
</Template>
</GroupFooter>

<GroupSeparator Tag="None">
<Template>
   </td></tr><tr><td>
</Template>
</GroupSeparator>

<DataRowSeparator>
   <NamedParts>
      <desDD:DataRowSeparatorNamedPatternPart Name="GroupAsMultipleColumns"
         Tag="None" UseStandardContainer="false">
      <Template>
         </div></td>
         <td><div<%= GetTagAttributes("StyleSheetName", "NamedStyle") %>>
      </Template>
      </desDD:DataRowSeparatorNamedPatternPart>
   </NamedParts>
</DataRowSeparator>

<%-- An empty box with the same styles as DataRow. --%>
<EmptyItem Tag="Div" NamedStyle="same as in <DataRow>"
   AlternativeNamedStyle="same as in <DataRow>"
   CssClass="same as in <DataRow>" AlternativeCssClass=" same as in <DataRow>" >
   <Template>
    
   </Template>
</EmptyItem>
```

3. Set these properties on DynamicListView:
   - **PatternTemplateName** = *desired PatternTemplate*
   - **EnableGrouping** = true
   - **GroupItemCount** = *the number of columns*
   - **ShowDataRowSeparator** = true

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

Adding the DynamicListView Class

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

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

   Visual Studio and Visual Web Developer Design Mode Users

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

   Text Entry Users

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

   ```
   <desDD:DynamicListView id="[YourControlID]" runat="server" />
   ```

   Programmatically creating the DynamicListView control

   - Identify the control which you will add the DynamicListView control to its **Controls** collection. Like all ASP.NET controls, the DynamicListView control can be added to any control that supports child controls, including Templates. If you are adding to a Page, typically this is added to the HtmlForms control.
   - Create an instance of the DynamicListView control class. The constructor takes no parameters.
   - Assign the **ID** property.
   - Add the DynamicListView control to the **Controls** collection.

   In this example, the DynamicListView is created with an **ID** of "DynamicListView1". It is added to PlaceHolder1.

   [C#]

   ```
   PeterBlum.DES.DynamicData.DynamicListView vDynamicListView =
      new PeterBlum.DES.DynamicData.DynamicListView();
   vDynamicListView.ID = "DynamicListView1";
   container.Controls.Add(vDynamicListView);
   ```

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

   [VB]

   ```
   Dim vDynamicListView As PeterBlum.DES.DynamicData.DynamicListView = _
      New PeterBlum.DES.DynamicData.DynamicListView()
   vDynamicListView.ID = "DynamicListView1"
   container.Controls.Add(vDynamicListView)
   ```

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

3. Your form should have a Dynamic Data compatible DataSource Control. Assign its ID to the **DataSourceID** property of the DynamicListView.
4. Your form should have a DynamicDataManager with a DynamicListViewAdapter object specified in its Adapters collection. The DynamicListViewAdapter's **DataBoundControlID** should specify the ID of your DynamicListView. See also "Properties common to all DataBound Control Adapters".

CONTINUED ON THE NEXT PAGE

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

5. Review the existing Pattern Templates to determine the right one. By default, Pattern Templates go in the **[web application]/DynamicData/PatternTemplates** folder.

   By default, **GridView.ascx** is used. It creates a columnar table with one row for each record.

   ListOfDetailViews.ascx provides a stacked appearance for fields, much like the DetailsView control.

   You may need to create one if none are a match. See "Customizing and creating your own PatternTemplates".

   Suggestion: Keep the PatterrnTemplate file open in an editor as you work.

6. If you are not using **GridView.ascx**, set the **PatternTemplateName** property to the file name of the Pattern Template file. The file extension (".ascx") is optional.

   If your Pattern Template file is not in the **[web application]/DynamicData/PatternTemplates** folder, set the URL to the correct folder within the **PatternTemplatesFolderUrl** property.

Guidelines for setting properties

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

7. Add the names of Data Fields in several ways:

   - Add `PeterBlum.DES.DynamicData.DataFieldInPattern` objects to the **ItemsInPattern** collection.

```
<desDD:DynamicListView id="DynamicListView1" runat="server"
    DataSourceID="DataSourceID" PatternTemplateName="MyPatternTemplate" >
    <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="columnname1" />
      <desDD:DataFieldInPattern DataField="columnname2" />
      <desDD:DataFieldInPattern DataField="columnname3" />
    </ItemsInPattern>
</desDD:DynamicListView>
```

   - Use automatic scaffolding by leaving the **ItemsInPattern** collection empty or by setting **AutoScaffoldDataFields** to `True`.

```
<desDD:DynamicListView id="DynamicListView1" runat="server"
    DataSourceID="DataSourceID" PatternTemplateName="MyPatternTemplate"
    AutoScaffoldDataFields="True" >
</desDD:DynamicListView>
```

   See "Defining a list of data fields".

8. You can add other objects to the **ItemsInPattern** collection to change the default formatting. See:

   - Buttons in PatternTemplates
   - Passing text into PatternTemplates
   - Using PatternTemplates to output content without data fields
   - Adding HTML and other web controls between data fields

CONTINUED ON THE NEXT PAGE

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

9. Determine the interface for buttons. By default, LinkButtons are generated as the first column.

   To switch to Buttons, set **ButtonsNamedPart** to "Buttons". If you have created another NamedPart for your buttons within the Pattern Template file, use its name. See "Buttons".

   To position the buttons in a different column, use the **AutoGenerateButtons** property. It defaults to `First`. Use `Last` or `FirstLast` to offer buttons in those columns. Use `None` if you will add a ButtonsInPattern object within the **ItemsInPattern** collection. This allows you to position them in any column.

10. Evaluate the formatting of the Pattern Template. If it needs adjustment to style sheets or specific style attributes, see if that element supports a Named Style. It will use a `<desDD:DynamicStylesTag>` control or a call to a method that supports Named Styles.

11. Add `PeterBlum.DES.DynamicData.NamedStyle` objects to the **NamedStyles** collection for each Named Style that you want to override.

    - Named Styles: Your pages control the formatting
    - NamedStyles on the GridView PatternTemplate
    - NamedStyles on the ListOfDetailViews PatternTemplate
    - The DynamicStylesTag Control
    - The NamedStyle Class

```
<desDD:DynamicListView id="DynamicListView1" runat="server"
  DataSourceID="DataSourceID" >
  <NamedStyles>
    <desDD:NamedStyle DataField="ProductName" Style="width:150px;" />
    <desDD:NamedStyle Name="SelectedRow" CssClass="SelectedRow" />
  </NamedStyles>
</desDD:DynamicListView>
```

12. Review the public properties exposed with the Pattern Template file's code.

    If there are any property values you want to change, see "Passing properties to be used by the PatternTemplate's own code".

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

Properties of the DynamicListView Class

The DynamicListView is subclassed from System.Web.UI.WebControls.ListView. Many properties from that class are not covered below. Use http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listview_members.aspx as a reference.

Click on any of these topics to jump to them:

- Data Properties
- PatternTemplate Properties
- Default NamedParts Properties
- Overall appearance Properties
- Header and Footer Properties
- Grouping Properties
- Template Properties
- Scaffold Caching Properties
- Behavior Properties
- Properties used when programming

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

Data Properties

These properties appear in the "Data" category of the Properties Editor.

- **DataSourceID** (string) – Gets or sets the ID of the control from which the data-bound control retrieves its list of data items. This must be a Data Source that is compatible with Dynamic Data, including LinqDataSource, EntityDataSource, and DynamicObjectDataSource.

  See "Establishing a DataSource".

- **DataMember** (string) – Generally you don't use this since the DataSource control should supply the correct data with its own query capabilities.

- **DataKeyNames** (array of strings) – Gets or sets an array that contains the names of the primary key fields for the items displayed in a DynamicListView control. See http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listview.datakeynames.aspx

- **AutoScaffoldDataFields** (enum PeterBlum.DES.DynamicData.AutoScaffoldDataFields) – Determines if the **ItemsInPattern** collection is automatically populated with Data Fields using the Automatic Scaffolding rules.

  When in use, this adds DataFieldInPattern objects to the ItemsInPattern collection by using the Scaffolding rules. When ItemsInPattern already has objects, scaffolded objects are added at the end of the collection.

  This property is ignored when the **PartsToGenerate** property does not reflect a data oriented part, such as `ContainerOpeningTag` and `Buttons`. It is also ignored when the **DataField** property is assigned.

  The enumerated type AutoScaffoldDataFields has these values:

  - `True` – Use automatic scaffolding to populate **ItemsInPattern**.
  - `False` – Do not modify the **ItemsInPattern** collection.
  - `WhenEmpty` - The **ItemsInPattern** collection must be empty to use automatic scaffolding. This is the default.

- **ItemsInPattern** (PeterBlum.DES.DynamicData.ItemsInPattern) – Use this collection to identify the data fields that will be assigned to DynamicControls found in the Pattern Template. It can also specify formatting available through NamedParts of PartsPatternTemplates and TemplateNotInPattern objects.

  See "Defining a list of data fields".

  The ItemInPattern classes are:

  - The DataFieldInPattern Class
  - The DataFieldListInPattern Class
  - The ScaffoldInPattern Class
  - The NoDataFieldsInPattern Class
  - The TemplateNotInPattern Class

  If you need to programmatically find a specific object within **ItemsInPattern**, there are two methods available on the DynamicListView control:

  - For DataFieldInPattern objects, use `FindByDataField("datafieldname")`.

    See "GetAttributeAsInteger() method".

  - For all objects, assign a unique value in the ItemInPattern object's ID property when the object is first created or in the ASP.NET markup. Then use `FindByID("id")` and typecast to the expected class.

    See "FindByID() method".

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

PatternTemplate Properties

These properties appear in the "Pattern Template" category of the Properties Editor.

- **PatternTemplateName** (string) – The name of the Pattern Template file that generates the content of this control. The ".ascx" extension is optional.

  The Pattern Template must be a PartsPatternTemplate.

  It defaults to `"GridView"`, which creates a table of rows for each record and columns for each data field.

  DES also provides **ListOfDetailViews.ascx**, which stacks fields much like the DetailsView control.

  If you assign this property to a file in a different folder from **[web application]/DynamicData/PatternTemplates**, set the URL to that folder in the **PatternTemplatesFolderUrl** property.

  For more, see "Choosing the PatternTemplate".

- **PatternTemplatesFolderUrl** (string) – The URL to the folder containing the Pattern Template file. If the file is in the **[web application]/DynamicData/PatternTemplates** folder, this property can be unassigned.

  It defaults to "".

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

Default NamedParts Properties

These properties appear in the "Default NamedParts" category of the Properties Editor.

Each Part in a PartsPatternTemplate can use a NamedPart instead of the part declared on the DynamicPartsPatterns control. The following chart indicates the properties that define a NamedPart by its name.

Part Type	Property name
Container	ContainerNamedPart
HeaderRow	HeaderRowNamedPart
HeaderCell	HeaderCellNamedPart **GridView.ascx** has a NamedPart that does not support sorting called "NoSort".
HeaderButtons	HeaderButtonsNamedPart
DataRow	DataRowNamedPart
DataCell	DataCellNamedPart
Buttons	ButtonsNamedPart Both **GridView.ascx** and **ListOfDetailViews.ascx** have NamedParts for LinkButtons and Standard buttons with these names: "LinkButtons" and "Buttons".
FooterRow	FooterRowNamedPart
FooterCell	FooterCellNamedPart
FooterButtons	FooterButtonsNamedPart
DataRowSeparator	DataRowSeparatorNamedPart
DataRowSeparatorCell	DataRowSeparatorCellNamedPart
NoDataToShow	NoDataToShowNamedPart
GroupContainer	GroupContainerNamedPart
GroupHeader	GroupHeaderNamedPart
GroupFooter	GroupFooterNamedPart
GroupSeparator	GroupSeparatorNamedPart
EmptyItem	EmptyItemNamedPart
EmptyItemCell	EmptyItemCellNamedPart

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

Overall appearance Properties

These properties appear in the "Overall appearance" category of the Properties Editor.

- **NamedStyles** (PeterBlum.DES.DynamicData.NamedStyles) – A list of NamedStyle objects. These are passed to all Pattern Templates generated within this DynamicListView control.

  Named Styles allow the user to deliver formatting rules into your Pattern Template. You declare various HTML tags that they can format with a DynamicStylesTag control, or by calling the `GetTagAttributes()` method. These take a name which is used to lookup a NamedStyle object from this list. (You don't have to search this list yourself.)

  For more details, see "Using Named Styles to adjust the formatting".

- **AutoGenerateButtons** (enum PeterBlum.DES.DynamicData.AutoGenerateButtons) – Determines if and where Buttons will appear on each row. The appearance for buttons is determined by the Pattern Template in **ButtonsNamedPart**.

  The enumerated type `PeterBlum.DES.DynamicData.AutoGenerateButtons` has these values:

  - `First` - first column. This is the default.
  - `Last` - last column.
  - `FirstLast` - both first and last.
  - `None` - Do not auto generate. Often the user declares a ButtonsInPattern object to generate buttons when set to `None`.

```
<desDD:DynamicListView id="DynamicListView1" runat="server"
   DataSourceID="RecordsListDataSource" AutoScaffoldDataFields="false"
   AutoGenerateButtons="None" >
   <ItemsInPattern>
      <desDD:ScaffoldInPattern />
      <desDD:ButtonsInPattern NamedPart="Buttons" />
   </ItemsInPattern>
</desDD:DynamicListView>
```

- **AlternatingRowStyles** (Boolean) – Determines if alternating rows in the body switch between two styles. The Pattern Template.DataRow and DataCell Parts offer style sheets and NamedStyles for Alternative as well as standard.

  Rows marked selected always apply the selected style instead of standard and alternative.

  When `true`, alternative rows use the alternative Named Styles unless the row is also selected.

  When `false`, all body rows use the standard Named Styles unless the row is selected.

  It defaults to `false`.

  The actual names for these Named Styles depends on the Pattern Template file, although **GridView.ascx** and **ListOfDetailViews.ascx** use these names: "DataRow", "AlternativeRow", and "SelectedRow".

- **ShowDataRowSeparator** (Boolean) – Determines if Data Rows are separated by the DataRow Separator. This Part is inserted between every pair of records (DataRows).

  The DataRow Separator is defined as the `<DataRowSeparator>` Part in the PartsPatternTemplate. It can either generate the complete HTML representing the separator, or a series of cells using the `<DataRowSeparatorCell>` Part.

  When only one record (DataRow) is shown on the page, no DataRow Separator is added. When used, it generates the **DynamicListView.ItemSeparatorTemplate** property's content. However, if **ItemSeparatorTemplate** is already assigned, it will not be replaced.

  When `true`, use the DataRow Separator.

  It defaults to `false`.

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

Header and Footer Properties

These properties appear in the "Header and Footer" category of the Properties Editor.

- **ShowHeaderRow** (Boolean) – Determines if the Header Row is added. The PartsPatternTemplate defines the HTML to output in its `<HeaderRow>`, `<HeaderCell>`, and `<HeaderButtons>` Parts. See "Header Row and Column Titles".

  When `true`, the Header Row is shown.

  When `false`, the Header Row is not shown.

  It defaults to `true`.

- **HeaderRules** (enum PeterBlum.DES.DynamicData.DynamicListViewHeaderRules) – Determines whether to use the automatically generated header row or **HeaderTemplate**. It can offer both and determine the order they are shown.

  The header row is only used when **ShowHeaderRow** is `true`.

  The enumerated type `PeterBlum.DES.DynamicData.DynamicListViewHeaderRules` has these values:

  - `Normal` - If **HeaderTemplate** is defined, use it. Otherwise use the automatic generated header row. This is the default.
  - `AutoOnly` - Only use the automatic generated header row.
  - `TemplateOnly` - Only use the **HeaderTemplate**. If not defined, use nothing.
  - `AutoThenTemplate` - First output the automatic generated header row. Then output the **HeaderTemplate**.
  - `TemplateThenAuto` - First output the **HeaderTemplate**. Then output the automatic generated header row.

- **HeaderTemplate** (ITemplate) – The controls and HTML that form a header row. It can be used together with or in place of the automatically generated header row based on the **HeaderRules** property.

  Use this to customize the header beyond what is available within the Pattern Template that format the column titles.

  Often used to create a table row on a list view.

  The header row is only used when **ShowHeaderRow** is `true`.

  For details, see "Header Row and Column Titles".

- **ShowFooterRow** (Boolean) – Determines if the footer row is added. The PartsPatternTemplate defines the HTML to output in its `<FooterRow>`, `<FooterCell>`, and `<FooterButtons>` Parts. See "Footer Row".

  When `true`, the footer row is added.

  When `false`, the footer row is not added.

  It defaults to `false`.

- **FooterRules** (enum PeterBlum.DES.DynamicData.DynamicListViewFooterRules) – Determines whether to use the automatically generated Footer row or **FooterTemplate**. It can offer both and determine the order they are shown.

  The Footer row is only used when **ShowFooterRow** is `true`.

  The enumerated type `PeterBlum.DES.DynamicData.DynamicListViewFooterRules` has these values:

  - `Normal` - If **FooterTemplate** is defined, use it. Otherwise use the automatic generated Footer row. This is the default.
  - `AutoOnly` - Only use the automatic generated Footer row.
  - `TemplateOnly` - Only use the **FooterTemplate**. If not defined, use nothing.
  - `AutoThenTemplate` - First output the automatic generated Footer row. Then output the **FooterTemplate**.
  - `TemplateThenAuto` - First output the **FooterTemplate**. Then output the automatic generated Footer row.

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

- **FooterTemplate** (ITemplate) – The controls and HTML that form a footer row. It can be used together with or in place of the automatically generated header row based on the **FooterRules** property.

  The footer row is only used when **ShowFooterRow** is `true`.

  For details, see “Footer Row”.

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

Grouping Properties

These properties appear in the "Grouping" category of the Properties Editor.

See "Grouping records to provide a tiled interface".

- **EnableGrouping** (Boolean) – Determines if the grouping feature is setup. It populates the **GroupTemplate** and **GroupSeparatorTemplate** properties if they are not already used.

  Grouping allows HTML around one or more complete items, with their own Container tag, header, footer, and body items. It helps establish formatting such as having items appear in different columns.

  Grouping is associated with these Parts in the PartsPatternTemplate:

  `<GroupContainer>` – Forms the HTML tag that encloses a group.

  `<GroupHeader>` – HTML between the opening tag and first item.

  `<GroupFooter>` – HTML between the last item and closing tag.

  `<GroupSeparator>` – HTML separating each group.

  `<EmptyItem>` and `<EmptyItemCell>` – HTML representing an item when there is no data to show for that data.

  Use the **GroupItemCount** property to determine how many items appear per group.

  When `true`, the grouping feature is setup.

  It defaults to `false`.

- **GroupItemCount** (integer) – The number of items within each group. See http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listview.groupitemcount.aspx.

- **GroupTemplate** (ITemplate) – Use the **GroupTemplate** property to create a tiled layout in the DynamicListView control. In a tiled table layout, the items are repeated horizontally in a row. The numbers of times that an item is repeated is specified by the **GroupItemCount** property. See http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listview.grouptemplate.aspx.

  To automatically generate it, set **EnableGrouping** to `true`. It will use the Parts described above in **EnableGrouping**.

  If **LayoutTemplate** is not assigned, make sure the **EnableGrouping** property is `true` so DynamicListView can generate it correctly for grouping.

- **GroupSeparatorTemplate** (ITemplate) – Adds custom HTML and web controls between each group. http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listview.groupseparatortemplate.aspx.

  When not assigned, it is automatically generated when **EnableGrouping** is `true`. It uses the `<GroupSeparator>` Part.

- **ItemSeparatorTemplate** (ITemplate) – Add custom HTML and web controls between each row of the body. See http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listview.itemseparatortemplate.aspx.

  When not assigned, it is automatically generated when **ShowDataRowSeparator** is `true`. It uses the `<DataRowSeparator>` Part.

- **EmptyItemTemplate** (ITemplate) – The empty item is displayed when there are no more data items to display in the last group of the current page. Add custom HTML and web controls for this area. See http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listview.emptyitemtemplate.aspx.

  When not assigned, it is automatically generated when **EnableGrouping** is `true`. It uses the `<EmptyItem>` and `<EmptyItemCell>` Parts.

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

Template Properties

- **ItemTemplate** (ITemplate) – A DataRow used when in read-only mode. When used, it replaces rows that are automatically generated using the `<DataRow>` Part within the PartsPatternTemplate. You must create all of the HTML and controls to do the work that was automatically generated. See "Using ListView's Templates instead of automatic generation".
- **EditItemTemplate** (ITemplate) – A DataRow used when in Edit mode. When used, it replaces rows that are automatically generated using the `<DataRow>` Part within the PartsPatternTemplate. You must create all of the HTML and controls to do the work that was automatically generated. See "Using ListView's Templates instead of automatic generation".
- **InsertItemTemplate** (ITemplate) – A DataRow used when in Insert mode. When used, it replaces rows that are automatically generated using the `<DataRow>` Part within the PartsPatternTemplate. You must create all of the HTML and controls to do the work that was automatically generated. See "Using ListView's Templates instead of automatic generation".
- **SelectedItemTemplate** (ITemplate) – A DataRow used when the row is selected (the SelectedIndex property identifies the row) and in read-only mode. When used, it replaces rows that are automatically generated using the `<DataRow>` tag within the PartsPatternTemplate. You must create all of the HTML and controls to do the work that was automatically generated. See "Using ListView's Templates instead of automatic generation".
- **AlternativeItemTemplate** (ITemplate) – A body row used in read-only mode for alternating rows. It is automatically generated when **AlternativeRowStyles** is `true`. When used, it replaces rows that are automatically generated using the `<DataRow>` tag within the PartsPatternTemplate. You must create all of the HTML and controls to do the work that was automatically generated. See "Using ListView's Templates instead of automatic generation".
- **LayoutTemplate** (ITemplate) – This template controls the overall structure of the control. If you elect to define this template, DynamicListView will no longer prepare this. Your Template should be setup with these elements in mind:
    - The opening HTML tag for the overall control.
    - A header row. If you want to use the automatic generator or **HeaderTemplate**, you must programmatically populate **LayoutTemplate**. Create a class that implements `System.Web.UI.WebControls.ITemplate`. Pass in the DynamicListView and call `DynamicListView.CreateHeaderRow(container)`.
    - A placeholder control for the items. Assign the ID of your PlaceHolder control to the **ItemPlaceHolderID** property. If you are using the **GroupTemplate** or **EnableGrouping** is `true`, instead use a placeholder control for the **GroupTemplate**. Assign the ID of your PlaceHolder control to the **GroupPlaceHolderID** property.
    - A footer row. If you want to use the automatic generator or **FooterTemplate**, you must programmatically populate **LayoutTemplate**. Create a class that implements `System.Web.UI.WebControls.ITemplate`. Pass in the DynamicListView and call `DynamicListView.CreateFooterRow(container)`.
    - The closing HTML tag for the overall control.
- **ItemSeparatorTemplate** (ITemplate) – Add custom HTML and web controls between each row of the body. See http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listview.itemseparatortemplate.aspx.

  When used, it replaces the content generated by the `<DataRowSeparator>` Part of the PartsPatternTemplate.
- **EmptyDataTemplate** (ITemplate) – Add HTML and web controls that appear when there are no rows of data available. See http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listview.emptydatatemplate.aspx**.**

  When used, it replaces the content generated by the `<NoDataToShow>` Part of the PartsPatternTemplate.

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

Scaffold Caching Properties

These properties appear in the "Scaffold Caching" category of the Properties Editor.

Automatic Scaffolding can cache the results of collecting a list of DataFieldInPattern objects to avoid generating the same list multiple times. These properties determine if scaffolding is used and how long data is preserved in the cache.

- **EnableScaffoldCache** (enum PeterBlum.DES.DynamicData.CacheLocation) – Determines if caching is used and its location when on.

  The user can turn off caching on an individual request simply by setting this to `None`.

  Cached data is assigned to a group with a named defined in **CacheCategoryName**. Separate cached data is kept by the MetaTable.Name requesting the data and by the data entry mode. The actual cache name combines all of these elements.

  The enumerated type CacheLocation has these values:

  - None - Cache is not used. Scaffolding is requested on each DynamicPattern control.
  - Page - Cache lasts for the page request. This allows many DynamicPatterns that work together to create a single data bound control to share a list of DataFieldInPattern objects. This is the default.
  - `User` - Cache per user using the Session class.
  - Global - Cache using the HttpCache class with a priority and timeout.

- **CacheCategoryName** (string) – Defines a category for this cache. Categories let you have multiple caches, for different situations.

  You might have a different dataset per User's role. This could be the role name.

  It defaults to "".

- **CacheExpirationSeconds** (integer) – Determines that entries cached in the cache expire after this many seconds.

  When in the Global cache, ASP.NET will automatically delete them when no longer needed. When in the User cache, they are only abandoned if requested and over this time limit. Not used when using the Page cache.

  It defaults to 60 (seconds).

- **CacheSlidingExpiration** (Boolean) – When using the Global or User cache, this determines if the expiration time (**CacheExpirationSeconds**) resets each time the item is retrieved from the cache.

  When `true`, expiration time resets each time the item is retrieved.

  When `false`, the expiration time is from the time initially added to the cache.

  It defaults to `false`.

- **GlobalCachePriority** (enum System.Web.Caching.CacheItemPriority) – When using the Global cache, this is the priority to preserve the data when the cache needs more memory.

  It defaults to `CacheItemPriority.Normal`.

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

Behavior Properties

- **Visible** (Boolean) – When `false`, this control is not used. It defaults to `true`.
- **AllowSelectingRows** (Boolean) – Determines if the user can select rows of the list. It establishes a Select button in the Pattern Template used for buttons and sets up the **SelectedItemTemplate** to format the row.

  It defaults to `false`.
- **AutoInsertItemMode** (enum PeterBlum.DES.DynamicData.AutoInsertItemMode) – The DynamicListView uses the **InsertItemPosition** property to determine if the **InsertItemTemplate** row is shown. **AutoInsertItemMode** determines how **InsertItemPosition** operates.

  The enumerated type `PeterBlum.DES.DynamicData.AutoInsertItemMode` has these values:

  - `No` - User must always use InsertItemPosition to show and hide this row.
  - `First` - The row is shown first in the list, unless another row is in Edit mode.
  - `Last` - The row is show last in the list, unless another row is in Edit mode.
  - `AutoDeactivate` - The user must set **InsertItemPosition** to activite Insert mode, but the grid does the work to remove the row when editing is finished. This is the default.

    Many actions finish editing: sorting, starting an edit row, selecting a row, etc.
- **InsertItemPosition** (enum System.Web.UI.WebControls.InsertItemPosition) – Determines if and where the InsertItemTemplate is shown to allow creating new records. Only set it when **AutoInsertItemMode** is `No` because otherwise, **AutoInsertItemMode** is managing this property for you. For details, see http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listview.insertitemposition.aspx.
- **InsertItemTemplateRequiresDataBinding** (Boolean) – The ListView doesn't normally call DataBind on an InsertItemTemplate. This forces it to be databound. It DataBinds by default. If you don't use databinding features in your Pattern Template's **InsertItemTemplate**, set this to `false` to improve performance.

  When `true`, databinding occurs automatically on InsertItemTemplate.

  When `false`, databinding is not applied to the InsertItemTemplate.

  It defaults to `true`.
- **EnableModelValidation** (Boolean) – Gets or sets a value that indicates whether a validator control will handle exceptions that occur during insert or update operations. While it defaults to `false`, by using it with Dynamic Data, the DynamicListViewAdapter will set it to `true` during a postback or callback. Generally you do not modify this property.
- **ItemPlaceholderID** (string) – When you replace the **LayoutTemplate** with your own, set this to the ID of a Placeholder control within your template where the body (item) rows will appear. Do not use it when you let DynamicListView generate the LayoutTemplate for you. For details, see http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listview.itemplaceholderid.aspx.
- **GroupPlaceholderID** (string) – When you replace the LayoutTemplate with your own, set this to the ID of the Placeholder control within your template where the **GroupTemplate** belongs. Do not use it when you let DynamicListView generate the **LayoutTemplate** for you. For details, see http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listview.groupplaceholderid.aspx.

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

- **AutoLoadPatternTemplateStyleSheetFile** (Boolean) – Pattern Templates often use the style sheet classes defined in the file: **~/DynamicData/PatternTemplateNamedStyles.css**. When this property is `true`, that file is automatically added to the page without you establishing a Link tag.

  When used, it merges the style sheet file with other DES files. It also compresses the file, removing comments and much whitespace.

  When not used, you can still load it merged and compressed by calling this in `Page_Load()`.

  ```
  PeterBlum.DES.Globals.Page.StyleSheetManager.Include(
      PeterBlum.DES.StyleSheetFiles.PatternTemplateNamedStyles)
  ```

  The path to this file can be changed in two ways:

  In `Application_Start()`, call

  ```
  PeterBlum.DES.StyleSheetManager.SetDefaultUrl(
      PeterBlum.DES.StyleSheetFiles.PatternTemplateNamedStyles, "URL")
  ```

  In the `<appSettings>` section of **web.config**, add

  ```
  <add key="DES_StyleSheetPatternTemplateNamedStyles" value="URL" />
  ```

  It autoloads when `true`.

  It defaults to `true`.

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

Properties used when programming

These properties are used when coding against the DynamicListView. They are not used in design mode.

- **EditIndex** (integer) – Gets or sets the index to the row that is in edit mode. (Only one row can be in edit mode at a time.) When -1, no row is in edit mode. Otherwise rows start at 0. See http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listview.editindex.aspx.
- **SelectedIndex** (integer) – Gets or sets the index of the selected item. When -1, no row is selected. Otherwise rows start at 0. See http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listview.selectedindex.aspx.
- **SelectedDataKey** (string) – Gets the data-key value for the selected item. See http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.listview.selecteddatakey.aspx.

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

The DynamicFormView Control

The DynamicFormView control is the recommended DataBound control for single record-style interfaces within DES Dynamic Data. While you can also use the DetailsView and FormView ontrols for these interfaces, both have limitations that are overcome by the DynamicFormView control.

Quality	DetailsView	FormView	DynamicFormView
Easy to setup	Very easy	Difficult	Very easy
Ability to customize HTML	Minimal	Complete	Complete
Supports automatic scaffolding	Yes	No	Yes

DynamicFormView is really a blend of the FormView control with Pattern Templates. It subclasses from `System.Web.IU.WebControls.FormView`. As a result, it has every feature of FormView. Internally, it populates all of the Templates found on FormView with Pattern Templates. Pattern Templates offload all formatting into separate files (in the **[web application]\DynamicData\PatternTemplates** folder.) You provide a list of data fields to the DynamicFormView (optionally using automatic scaffolding) and it builds the HTML output by connecting each data field name to the controls defined in the Pattern Template.

If needed, you can switch to declaring controls directly in its Templates, making it return to its FormView roots.

The default DynamicFormView control uses the **DetailsView.ascx** Pattern Template file which mimics the appearance of the DetailsView control.

Click on any of these topics to jump to them:

- Using the DynamicFormView Class
- Adding the DynamicFormView Class
- Properties of the DynamicFormView Class


```
<desDD:DynamicFormView ID="DynamicFormView1" runat="server"
  DataSourceID="ListDataSource" >
</desDD:DynamicFormView>

<desDD:DynamicFormView ID="DynamicFormView1" runat="server"
  DataSourceID="ListDataSource" DataKeyNames="ProductID" >
  <ItemsInPattern>
    <desDD:DataFieldInPattern DataField="ProductName"  />
    <desDD:DataFieldInPattern DataField="UnitPrice"  />
    <desDD:DataFieldInPattern DataField="QuantityPerUnit"  />
    <desDD:DataFieldInPattern DataField="ReorderLevel"  />
    <desDD:DataFieldInPattern DataField="UnitsInStock"  />
    <desDD:DataFieldInPattern DataField="Discontinued"  />
  </ItemsInPattern>
  <NamedStyles>
    <desDD:NamedStyle DataField="QuantityPerUnit" Style="white-space:nowrap;" />
  </NamedStyles>
</desDD:DynamicFormView>
```

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

Using the DynamicFormView Class

Like all DataBound controls, your web form must contain a Dynamic Data compatible Data Source and a DynamicDataManager control with a DataBound Control Adapter that identifies your DynamicFormView control. If the list supports editing, also add a ValidationSummary control.

Here is the basic markup.

```
<desDD:DynamicDataManager ID="DynamicDataManager1" runat="server" >
   <Adapters>
      <desDD:DynamicFormViewAdapter DataBoundControlID="DynamicFormView1" />
   </Adapters>
</desDD:DynamicDataManager>

<desDD:DynamicFormView ID="DynamicFormView1" runat="server"
   DataSourceID="ListDataSource" >
</desDD:DynamicFormView>

<asp:LinqDataSource ID="ListDataSource" runat="server"
   ContextTypeName="DataContext" TableName="tablename"
   EnableUpdate="boolean" EnableInsert="boolean" EnableDelete="boolean">
</asp:LinqDataSource>
```

This example adds editing capabilities. Both the DataSource and Adapter have new properties, plus the ValidationSummary has been added.

```
<desDD:DynamicDataManager ID="DynamicDataManager1" runat="server" >
   <Adapters>
      <desDD:DynamicFormViewAdapter DataBoundControlID="DynamicFormView1"
         SupportsEditMode="true" ValidationSummaryControlID="ValidationSummary1" />
   </Adapters>
</desDD:DynamicDataManager>

<des:ValidationSummary ID="ValidationSummary1" runat="server" InAJAXUpdate="True"/>

<desDD:DynamicFormView ID="DynamicFormView1" runat="server"
   DataSourceID="ListDataSource" >
</desDD:DynamicFormView>

<asp:LinqDataSource ID="ListDataSource" runat="server"
   ContextTypeName="DataContext" TableName="tablename" EnableUpdate="true" >
</asp:LinqDataSource>
```

Click on any of these topics to jump to them:

- Comparing to FormView
- How to use the PartsPatternTemplate with a DynamicFormView
- Parts Overview for PartsPatternTemplate
- Establishing a DataSource
- Defining a list of data fields
- Changing the appearance
- Passing properties to be used by the PatternTemplate's own code
- Offering Paging

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

Comparing to FormView

As a direct subclass of the FormView control, DynamicFormView supports most of the capabilities describing FormView. Here are some sources of information:

- Control overview
- Class overview

There are exceptions because this control is designed to work within Dynamic Data. Here are some issues to be aware of:

- The Pattern Template file must inherit from `PeterBlum.DES.DynamicData.PartsPatternTemplate` as this Pattern Template supplies all of the parts that make up a list of fields.
- The DataSource must support Dynamic Data, such as LinqDataSource, EntityDataSource, and DynamicObjectDataSource.
- Most template properties are populated automatically, following the rules of the **ItemsInPattern** and **NamedStyles** collections. If you use the template properties directly, your template code will be used instead of the automatic code.

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

How to use the PartsPatternTemplate with a DynamicFormView

DynamicFormView demands that you select a Pattern Template file which inherits from `PeterBlum.DES.DynamicData.PartsPatternTemplate`. The PartsPatternTemplate supplies all of the parts that make up a list of fields that show a single record.

Explore the **DetailsView.ascx** Pattern Template file in the **[web application]/DynamicData/PatternTemplates** folder to see how it defines the output. The overall structure for these files looks like this:

```
<%@ Control Language="C#" Inherits="PeterBlum.DES.DynamicData.PartsPatternTemplate"
      Usage="PartsForSingle" %>
<%@ Register tagprefix="desDD" assembly="PeterBlum.DES.DynamicData"
      namespace="PeterBlum.DES.DynamicData" %>

<script runat="server">
   public override DynamicPartsPatterns GetDynamicPartsPatterns()
   {
      return DynamicPartsPatterns1;
   }
</script>

<desDD:DynamicPartsPatterns ID="DynamicPartsPatterns1" runat="server">

      <Various Parts and optional NamedParts>

</desDD:DynamicPartsPatterns>
```

You generally edit the inner content of the DynamicPartsPatterns control by adding or modifying Parts tags. Here are a few examples taken from **DetailsView.ascx**.

```
<DataRow Tag="None"/>

<DataCell Tag="Tr" NamedStyle="DataFieldRow" CssClass="<%#cCssDataFieldRow %>" >
  <Template>
     <desDD:DynamicStylesTag runat="server" Tag="Td"
        CssClass="<%# cCssLabelColumn %>" NamedStyle="LabelColumn"
        Style="vertical-align:top;" >

        <desDD:DynamicLabel ID="DynamicLabel1" runat="server"
           AssociatedControlID="DynamicControl1" />

     </desDD:DynamicStylesTag>

     <desDD:DynamicStylesTag runat="server" Tag="Td"
        CssClass="<%# cCssDataColumn %>" NamedStyle="DataColumn"
        Style="vertical-align:top;" >

        <desDD:DynamicControl ID="DynamicControl1" runat="server" />

     </desDD:DynamicStylesTag>

   </Template>
   <NamedParts>
      <Various NamedParts>
   </NamedParts>

</DataCell>

<NoDataToShow Tag="Div" NamedStyle="NoDataToShow"
      CssClass="<%# cCssNoDataToShow %>" Message="No records were found." >
</NoDataToShow>
```

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

Parts Overview for PartsPatternTemplate when used with Single Records

Here is a quick overview of the Parts where you define the HTML output.

Part tag	Description	In Template
`<Container>`	The HTML tag around the entire control. Specify its **Tag** property to define the HTML tag. It does not supply a Template property for any of its inner content because the rest of the tags are the inner content.	HeaderTemplate and FooterTemplate
`<DataRow>`	Describes the content for a single record. This Part tag provides three templates that define different output based on the data entry mode. You can declare the DynamicControls here, but if you want them to be generated for each data field automatically, use the <DataCell> for that and use <DataRow> to define the HTML tag that encloses the list of <DataCell> parts.	All ItemTemplates, including Edit, and Insert
`<DataCell>`	Describes the content of a single field in the record. This Part tag provides three templates that define different output based on the data entry mode. It should have at least one DynamicControl. It is used when there is no Template defined in <DataRow> or if the Template includes a PlaceHolder for the <DataCell> parts.	Depends on <DataRow>
`<Buttons>`	Describes the element that shows buttons for the record. The buttons depend on the data entry mode, such as New, Edit, and Details Buttons for ReadOnly mode and Save and Cancel Buttons for Insert and Edit modes. This part often contains a DynamicButtons control which knows the correct buttons to show based on the mode and various other settings in your form. This part is only used when the <DataRow> also allows the output of <DataCells>. It depends on either the **DynamicFormView.AutoGenerateButtons** property or the presence of a ButtonsInPattern object in the **ItemsInPattern** collection.	Depends on <DataRow>
`<HeaderRow>`	Describes the content for a header prior to the list of records. It often includes column titles. Requires the **DynamicFormView.ShowHeaderRow** to be `true`.	HeaderTemplate
`<HeaderCell>`	Not recommended for DynamicFormView.	
`<HeaderButtons>`	Not recommended for DynamicFormView.	

CONTINUED ON THE NEXT PAGE

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

Part tag	Description	In Template
`<FooterRow>`	Describes the content for a footer after to the list of records. Requires the **DynamicFormView.ShowFooterRow** to be `true`.	FooterTemplate
`<FooterCell>`	Not recommended for DynamicFormView.	
`<FooterButtons>`	Not recommended for DynamicFormView.	
`<DataRowSeparator>`	Not supported by DynamicFormView.	
`<DataRow-SeparatorCell>`	Not supported by DynamicFormView.	
`<NoDataToShow>`	Used when there is no data found. It is often a message like "No data found."	EmptyDataTemplate
`<GroupContainer>`	Not supported by DynamicFormView.	
`<GroupHeader>`	Not supported by DynamicFormView.	
`<GroupFooter>`	Not supported by DynamicFormView.	
`<GroupSeparator>`	Not supported by DynamicFormView.	
`<EmptyItem>`	Not supported by DynamicFormView.	
`<EmptyItemCell>`	Not supported by DynamicFormView.	

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

Establishing a DataSource

Always set the **DataSourceID** property to the ID of a DataSource control that is compatible with Dynamic Data. The DataSource control is expected to be in the same naming container as the DynamicFormView control.

Establishing Edit mode

If DynamicFormView should offer editing capabilities, set the **EnableUpdate** property to `true` in the DataSource and **SupportsEditMode** property to `true` on the DynamicFormViewAdapter object.

Establishing Insert mode

If DynamicFormView should offer inserting capabilities, set the **EnableInsert** property to `true` in the DataSource and **SupportsInsertMode** property to `true` on the DynamicFormViewAdapter object.

Establishing Row Deletion

If DynamicFormView should offer a Delete button, set the **EnableDelete** property to `true` in the DataSource.

```
<desDD:DynamicDataManager ID="DynamicDataManager1" runat="server" >
   <Adapters>
      <desDD:DynamicFormViewAdapter DataBoundControlID="DynamicFormView1"
         SupportsEditMode="boolean" SupportsInsertMode="boolean" />
   </Adapters>
</desDD:DynamicDataManager>

<desDD:DynamicFormView ID="DynamicFormView1" runat="server"
   DataSourceID="ListDataSource" AutoGenerateFields="true" >
</desDD:DynamicFormView>

<asp:LinqDataSource ID="ListDataSource" runat="server"
   ContextTypeName="DataContext" TableName="tablename"
   EnableUpdate="boolean" EnableInsert="boolean" EnableDelete="boolean">
</asp:LinqDataSource>
```

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

Defining a list of data fields

The DynamicFormView displays data from a data source. The data source provides it with the data – the table and rows requested by a query. The DynamicFormView needs you to determine which data fields are displayed and the order. There are two ways to approach this:

- Use Automatic Scaffolding. If you leave the **ItemsInPattern** collection empty, scaffolding is used unless you set **AutoScaffoldDataFields** to `False`. If you have items in **ItemsInPattern**, set the **AutoScaffoldDataFields** property to `True`. See "Automatic Scaffolding Overview".

```
<desDD:DynamicFormView ID="DynamicFormView1" runat="server"
  DataSourceID="RecordDataSource" >
</desDD:DynamicFormView>
```

```
<desDD:DynamicFormView ID="DynamicFormView1" runat="server"
  DataSourceID="FormDataSource" AutoScaffoldDataFields="True" >
  <ItemsInPattern>
    <desDD:NoDataInPattern NamedPart="Logo"  />
  </ItemsInPattern>
</desDD:DynamicFormView>
```

- Populate the **ItemsInPattern** collection with DataFieldInPattern and other ItemInPattern objects.

```
<desDD:DynamicFormView ID="DynamicFormView1" runat="server"
  DataSourceID="RecordDataSource" >
  <ItemsInPattern>
    <desDD:DataFieldInPattern DataField="ProductName"  />
    <desDD:DataFieldInPattern DataField="UnitPrice"  />
    <desDD:DataFieldInPattern DataField="QuantityPerUnit"  />
    <desDD:DataFieldInPattern DataField="ReorderLevel"  />
    <desDD:DataFieldInPattern DataField="UnitsInStock"  />
    <desDD:DataFieldInPattern DataField="Discontinued"  />
  </ItemsInPattern>
</desDD:DynamicFormView>
```

See the following topics:

- Defining ItemInPattern objects in the ItemsInPattern collection
- Defining the Data Field names shown by the PatternTemplate
- Adding HTML and other web controls between data fields
- Passing properties to the Field Templates

The ItemInPattern classes are:

- The DataFieldInPattern Class
- The DataFieldListInPattern Class
- The ScaffoldInPattern Class
- The ButtonsInPattern Class
- The LabelInPattern Class
- The NoDataFieldsInPattern Class
- The TemplateNotInPattern Class

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

Changing the appearance

Most of the interface comes from the Pattern Template files. They possess default style sheet classes and HTML formatting rules. Pattern Templates can be customized by editing them. However, that may change a Pattern Template that is correctly formatted for another webform's usage. Instead consider these approaches.

Click on any of these topics to jump to them:

- Choosing the PatternTemplate
- Using Named Styles to adjust the formatting
- Buttons
- Header Row and Column Titles
- Footer Row

Choosing the PatternTemplate

The DynamicFormView works with any Pattern Template that subclasses from `PeterBlum.DES.DynamicData.PartsPatternTemplate`. It ships with the **DetailsView.ascx** Pattern Template, which mimics the appearance of a DetailsView control. If you want to change from the default, set its name in the **PatternTemplateName** property.

- Using DetailsView.ascx PatternTemplate
- Customizing and creating your own PatternTemplates

If you want to change an existing Pattern Template file, consider making a new file because you may break web forms already using the Pattern Template file. Duplicate the Pattern Template file and edit the duplicate. Assign the file name of the new file in the **PatternTemplateName** property. The file extension ".ascx" is optional.

```
<desDD:DynamicFormView ID="DynamicFormView1" runat="server"
   DataSourceID="ListDataSource" PatternTemplateName="DetailsViewStyle2" >
</desDD:DynamicFormView>
```

If you created the file in a different folder than **[web application]/DynamicData/PatternTemplates**, use the **PatternTemplatesFolderUrl** property to specify the path to that folder.

The DynamicListView control offers more parts than you may want. It offers these properties to use or omit the parts:

- **ShowHeaderRow** – Shows the `<HeaderRow>` Part. It is hidden by default.
- **ShowFooterRow** – Shows the `<FooterRow>` Part. It is hidden by default.
- **AutoGenerateButtons** – Determines if the `<Buttons>` part is added either before or after the data fields. By default, it is set to `Last` to generate items before.

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

Using Named Styles to adjust the formatting

Named Styles let you override the style sheet and other formatting rules on individual elements within the Pattern Template without modifying the Pattern Template. Instead, you declare a list of NamedStyle objects in the **NamedStyles** collection of the DynamicFormView control.

Evaluate the Pattern Template file you have selected to see if you can use Named Styles. The Pattern Template files shipped with DES Dynamic Data all supply numerous Named Styles.

```
<desDD:DynamicFormView ID="DynamicFormView1" runat="server"
   DataSourceID="RecordDataSource" >
   <NamedStyles>
      <desDD:NamedStyle Name="LabelControl" CssClass="SmallLabel" />
      <desDD:NamedStyle Name="Row" Category="Title" CssClass="Title" />
      <desDD:NamedStyle Name="Row" DataField="QuantityPerUnit"
           Style="white-space:nowrap;" />
   </NamedStyles>
</desDD:DynamicFormView>
```

Because a NamedStyle can evaluate data field, category, and user role names, you may specify several NamedStyles with the same name, but different combinations of the other evaluators. It will always use the most specific match over a less specific match. For example, in the above example, it uses the third NamedStyle when "DataColumn" is being generated. If that is not generating output for "QuantityPerUnit", it then uses the second NamedStyle. If two NamedStyles could match and have the same number of evaluators, it uses the order specified in the **NamedStyles** property.

See the following topics:

- Named Styles: Your pages control the formatting
- NamedStyles on the DetailsView PatternTemplate
- NamedStyles on these PatternTemplate
- The DynamicStylesTag Control
- The NamedStyle Class

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

Buttons

By default, the **DetailsView.ascx** Pattern Template uses LinkButtons for buttons. If you want to change to standard Buttons, set the **ButtonsNamedPart** property to "`Buttons`".

```
<desDD:DynamicFormView ID="DynamicFormView1" runat="server"
   DataSourceID="RecordDataSource" ButtonsNamedPart="Buttons" >
</desDD:DynamicFormView>
```

Buttons appear in the last row by default. To position the buttons in a different column, use the **AutoGenerateButtons** property. It defaults to `Last`. Use `First` or `FirstLast` to offer buttons in those columns. Use `None` if you will add a ButtonsInPattern object whose **NamedPart** property specifies your interface for buttons within the **ItemsInPattern** collection. This allows you to position them in any column.

If the predefined appearances for buttons don't match your requirements, you can create another interface in the `<Buttons>` Part of the PartsPatternTemplate.

Buttons get their formatting through their own Pattern Template files. DES Dynamic Data includes these files for Button and LinkButton output: **DetailsViewButtons.ascx**, **DetailsViewLinkButtons.ascx**, **ButtonsForSingle.ascx**, and **LinkButtonsForSingle.ascx**. The **DetailsView.ascx** Pattern Template declares two NamedParts that map to these files. They are "`Buttons`" and "`LinkButtons`".

```
<Buttons PatternTemplateName="DetailsViewLinkButtons" other properties >
   <NamedParts>
      <desDD:ButtonsNamedPatternPart Name="LinkButtons"
        PatternTemplateName="DetailsViewLinkButtons" />
      <desDD:ButtonsNamedPatternPart Name="Buttons"
        PatternTemplateName="DetailsViewButtons" />
   </NamedParts>
</Buttons>
```

To use different formatting with Pattern Templates.

1. Duplicate one of the Button Pattern Template files and edit it.
2. Add a NamedPart into the `<Buttons>` Part of the PartsPatternTemplate:

```
<Buttons PatternTemplateName="DetailsViewLinkButtons" other properties >
   <NamedParts>
      <desDD:ButtonsNamedPatternPart Name="LinkButtons"
        PatternTemplateName="DetailsViewLinkButtons" />
      <desDD:ButtonsNamedPatternPart Name="Buttons"
        PatternTemplateName="DetailsViewButtons" />
      <desDD:ButtonsNamedPatternPart Name="newname"
        PatternTemplateName="newPatternTemplate" />
   </NamedParts>
</Buttons>
```

You can also add a DynamicButtons control to the Template property of the `<Buttons>` Part or one of its NamedParts.

```
<Buttons PatternTemplateName="DetailsViewLinkButtons" other properties >
   <NamedParts>
      <desDD:ButtonsNamedPatternPart Name="LinkButtons"
        PatternTemplateName="DetailsViewLinkButtons" />
      <desDD:ButtonsNamedPatternPart Name="Buttons"
        PatternTemplateName="DetailsViewButtons" />
      <desDD:ButtonsNamedPatternPart Name="newname">
         <Template>
            <desDD:DynamicButtons id="DynamicButtons1" runat="server" properties />
         </Template>
      </desDD:ButtonsNamedPatternPart>
   </NamedParts>
</Buttons>
```

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

Header Row and Column Titles

Note: The PartsPatternTemplate file may permanently disable the Header by setting the Enabled property to false within its <HeaderRow> Part.

The header row is generally used to show column titles. It is optional and hidden by default. If you want to show it, set **ShowHeaderRow** to `true`.

The formatting of the header row is determined by the `<HeaderRow>` Part within the Pattern Template file. **DetailsView.ascx** creates a two column table row in its <HeaderRow>. Its titles default to "Name" and "Value". You can modify them by using these property names in the DynamicFormView:

- **HeaderLabelColumnHTML** (string) – Text in the first column of the header.
- **HeaderDataColumnHTML** (string) – Text in the second column of the header.

To set properties declared on a Pattern Template, either define them directly in the ASP.NET markup of the DynamicFormView, or programmatically set them with the `SetAttributeAsNative()` method.

```
<desDD:DynamicFormView ID="DynamicFormView1" runat="server"
   DataSourceID="RecordDataSource" HeaderLabelColumnHTML="<b>Name</b>" >
</desDD:DynamicFormView>
```

```
DynamicFormView1.SetAttributeAsNative("HeaderLabelColumnHTML", "<b>Name</b>")
```

See "Passing properties to be used by the PatternTemplate's own code".

Footer Row

Note: The PartsPatternTemplate file may permanently disable the Footer by setting the Enabled property to false within its <FooterRow> Part.

The Footer row is generally used to show column titles. It is optional and hidden by default. If you want to show it, set **ShowFooterRow** to `true`.

The formatting of the Footer row is determined by the `<FooterRow>` Part within the Pattern Template file. **DetailsView.ascx** creates a two column table row in its <FooterRow>. Its titles both default to "` `". You can modify them by using these property names in the DynamicFormView:

- **FooterLabelColumnHTML** (string) – Text in the first column of the Footer.
- **FooterDataColumnHTML** (string) – Text in the second column of the Footer.

To set properties declared on a Pattern Template, either define them directly in the ASP.NET markup of the DynamicFormView, or programmatically set them with the `SetAttributeAsNative()` method.

```
<desDD:DynamicFormView ID="DynamicFormView1" runat="server"
   DataSourceID="RecordDataSource" FooterLabelColumnHTML="<hr />" >
</desDD:DynamicFormView>
```

```
DynamicFormView1.SetAttributeAsNative("FooterLabelColumnHTML", "<hr />")
```

See "Passing properties to be used by the PatternTemplate's own code".

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

Using FormView's Templates instead of automatic generation

DynamicFormView has numerous Template properties to generate its parts. When they are not assigned, DynamicFormView generates their content for you, by using a DynamicPattern control and the PartsPatternTemplate file. If you want to customize the Template directly, assign the desired HTML and web controls to it. Most use this to generate content from the PartsPatternTemplate:

```
<TemplateName>
  <desDD:DynamicPattern id="id" runat="server"
    PatternTemplateName="patterntemplatename" PartsToGenerate="PartsToGenerate" >
  </desDD:DynamicPattern>
</TemplateName>
```

It's easy to start your own markup with the markup that these controls would generate automatically.

3. Open the web form in Visual Studio and switch to Design mode.
4. Click on the SmartTag button.
5. Choose the appropriate the **Manage Auto Generation of Templates** command.
6. From the list, change templates from "Auto Generate" to "From Template" or back.

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

Passing properties to be used by the PatternTemplate's own code

Pattern Template can expose public properties that let you customize their behavior. For example, the **DetailsView.ascx** Pattern Template has the Boolean property **HeaderLabelColumnHTML** which specifies the text of the label in the HeaderRow above the Label column.

[C#]

```
public string HeaderLabelColumnHTML
{
   get { return Host.GetAttributeAsString("HeaderLabelColumnHTML",
      cDefaultHeaderLabelColumnHTML); }
}
private const string cDefaultHeaderLabelColumnHTML = "Name";
```

[VB]

```
Public ReadOnly Property HeaderLabelColumnHTML() As String
   Get
      Return Host.GetAttributeAsString("HeaderLabelColumnHTML",
         cDefaultHeaderLabelColumnHTML)
   End Get
End Property
Private Const cDefaultHeaderLabelColumnHTML As String = "Name"
```

The value of this property is assigned on the DynamicFormView control and retrieved by your property like this:

```
<desDD:DynamicFormView id="DynamicFormView1" runat="server"
   PatternTemplateName="DetailsView" HeaderLabelColumnHTML="Field Name" />
```

Just declare the property name and value within the ASP.NET markup of the DynamicFormView control. Some of the properties it already knows about and lists in the Properties Editor of Visual Studio. Others – especially those you created – are not known but still can be specified.

When programmatically assigning this property, you must pass it through the `Add()` method of the **Attributes** property:

```
DynamicFormView1.Attributes.Add("HeaderLabelColumnHTML", "Field Name")
```

Passing Properties specific to an ItemInPattern object

When using a PartsPatternTemplate, these properties can be passed along on individual DataFieldInPattern, DataFieldListInPattern, ButtonsInPattern, NoDataFieldsInPattern, and TemplateNotInPattern objects to be used by code within the Pattern Template in their **OnPartCreating** or **OnPartCreated** event handler methods.

The Pattern Template must call the `GetAttributeAsNative()` or `GetAttributeAsString()` methods on the *Args* parameter object passed to the **OnPartCreating** or **OnPartCreated** event handlers. Good documentation on the property names it supports is vital.

When the page developer identifies such a property, they specify the property name and value within the ASP.NET markup of the *xyz*InPattern object. In this example, the "ColumnTitle" NamedPart supports a property named "Title" that takes a string.

```
<desDD:DynamicFormView id="DynamicFormView1" runat="server"
   PatternTemplateName="MyPatternTemplate" >
   <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="columnname1" UIHint="FancyText"
         NamedPart="ColumnTitle" Title="my title" />
   </ItemsInPattern>
</desDD:DynamicFormView>
```

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

Offering Paging

Paging is sometimes used for single record views, especially those that are not editing data, to navigate through records. Use the DynamicControlPanel control to create a wide variety of paging interfaces. Place the control above or below the DynamicFormView as needed. You can have several on the page. Set the DynamicControlPanel's **PagedControlID** property assigned to the ID of the DynamicFormView.

Here is the DynamicFormView with DynamicControlPanel:

```
<desDD:DynamicFormView ID="RecordView" runat="server"
   DataSourceID="RecordDataSource" >
</desDD:DynamicFormView>
<desDD:DynamicControlPanel ID="DynamicControlPanel1" runat="server"
   PagedControlID="RecordView" >
   <Fields>
      <desDD:NextPreviousPagerField />
   </Fields>
</desDD:DynamicControlPanel>
```

See "The DynamicControlPanel Control".

Note: The ListView class offers its own Paging features by using the AllowPaging, PagerSettings, PagerStyle, and PagerTemplate properties. They can be used as well. The DynamicControlPanel control has a richer toolset.

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

Adding the DynamicFormView Class

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

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

   Visual Studio and Visual Web Developer Design Mode Users

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

   Text Entry Users

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

   ```
   <desDD:DynamicFormView id="[YourControlID]" runat="server" />
   ```

   Programmatically creating the DynamicFormView control

   - Identify the control which you will add the DynamicFormView control to its **Controls** collection. Like all ASP.NET controls, the DynamicFormView control can be added to any control that supports child controls, including Templates. If you are adding to a Page, typically this is added to the HtmlForms control.
   - Create an instance of the DynamicFormView control class. The constructor takes no parameters.
   - Assign the **ID** property.
   - Add the DynamicFormView control to the **Controls** collection.

   In this example, the DynamicFormView is created with an **ID** of "DynamicFormView1". It is added to PlaceHolder1.

   [C#]

   ```
   PeterBlum.DES.DynamicData.DynamicFormView vDynamicFormView =
      new PeterBlum.DES.DynamicData.DynamicFormView();
   vDynamicFormView.ID = "DynamicFormView1";
   container.Controls.Add(vDynamicFormView);
   ```

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

   [VB]

   ```
   Dim vDynamicFormView As PeterBlum.DES.DynamicData.DynamicFormView = _
      New PeterBlum.DES.DynamicData.DynamicFormView()
   vDynamicFormView.ID = "DynamicFormView1"
   container.Controls.Add(vDynamicFormView)
   ```

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

3. Your form should have a Dynamic Data compatible DataSource Control. Assign its ID to the **DataSourceID** property of the DynamicFormView.
4. Your form should have a DynamicDataManager with a DynamicFormViewAdapter object specified in its Adapters collection. The DynamicFormViewAdapter's **DataBoundControlID** should specify the ID of your DynamicFormView. See also "Properties common to all DataBound Control Adapters".

CONTINUED ON THE NEXT PAGE

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

5. Review the existing Pattern Templates to determine the right one. By default, Pattern Templates go in the **[web application]/DynamicData/PatternTemplates** folder.

   By default, **DetailsView.ascx** is used. It creates a row for each data field with a column for the label and another for the value.

   You may need to create one if none are a match. See "Customizing and creating your own PatternTemplates".

   Suggestion: Keep the PatterrnTemplate file open in an editor as you work.

6. If you are not using **DetailsView.ascx**, set the **PatternTemplateName** property to the file name of the Pattern Template file. The file extension (".ascx") is optional.

   If your Pattern Template file is not in the **[web application]/DynamicData/PatternTemplates** folder, set the URL to the correct folder within the **PatternTemplatesFolderUrl** property.

Guidelines for setting properties

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

7. Add the names of Data Fields in several ways:

   - Add `PeterBlum.DES.DynamicData.DataFieldInPattern` objects to the **ItemsInPattern** collection.

```
<desDD:DynamicFormView id="DynamicFormView1" runat="server"
    DataSourceID="DataSourceID" PatternTemplateName="MyPatternTemplate" >
    <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="columnname1" />
      <desDD:DataFieldInPattern DataField="columnname2" />
      <desDD:DataFieldInPattern DataField="columnname3" />
    </ItemsInPattern>
</desDD:DynamicFormView>
```

   - Use automatic scaffolding by leaving the **ItemsInPattern** collection empty or by setting **AutoScaffoldDataFields** to `True`.

```
<desDD:DynamicFormView id="DynamicFormView1" runat="server"
    DataSourceID="DataSourceID" PatternTemplateName="MyPatternTemplate"
    AutoScaffoldDataFields="True" >
</desDD:DynamicFormView>
```

   See "Defining a list of data fields".

8. You can add other objects to the **ItemsInPattern** collection to change the default formatting. See:

   - Buttons in PatternTemplates
   - Passing text into PatternTemplates
   - Using PatternTemplates to output content without data fields
   - Adding HTML and other web controls between data fields

CONTINUED ON THE NEXT PAGE

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

9. Determine the interface for buttons. By default, LinkButtons are generated as the first column.

   To switch to Buttons, set **ButtonsNamedPart** to "Buttons". If you have created another NamedPart for your buttons within the Pattern Template file, use its name. See "Buttons".

   To position the buttons in a different column, use the **AutoGenerateButtons** property. It defaults to `Last`. Use `First` or `FirstLast` to offer buttons in those columns. Use `None` if you will add a ButtonsInPattern object whose **NamedPart** property specifies your interface for buttons within the **ItemsInPattern** collection. This allows you to position them in any column.

10. Evaluate the formatting of the Pattern Template. If it needs adjustment to style sheets or specific style attributes, see if that element supports a Named Style. It will use a `<desDD:DynamicStylesTag>` control or a call to a method that supports Named Styles.

11. Add `PeterBlum.DES.DynamicData.NamedStyle` objects to the **NamedStyles** collection for each Named Style that you want to override.

    - Named Styles: Your pages control the formatting
    - NamedStyles on the DetailsView PatternTemplate
    - The DynamicStylesTag Control
    - The NamedStyle Class

```
<desDD:DynamicFormView id="DynamicFormView1" runat="server"
  DataSourceID="DataSourceID" >
  <NamedStyles>
    <desDD:NamedStyle DataField="ProductName" Style="width:150px;" />
    <desDD:NamedStyle Name="SelectedRow" CssClass="SelectedRow" />
  </NamedStyles>
</desDD:DynamicFormView>
```

12. Review the public properties exposed with the Pattern Template file's code.

    If there are any property values you want to change, see "Passing properties to be used by the PatternTemplate's own code".

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

Properties of the DynamicFormView Class

The DynamicFormView is subclassed from `System.Web.UI.WebControls.FormView`. Many properties from that class are not covered below. Use http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.formview_members.aspx as a reference.

Click on any of these topics to jump to them:

- Data Properties
- PatternTemplate Properties
- Default NamedParts Properties
- Appearance Properties
- Template Properties
- Scaffold Caching Properties
- Behavior Properties

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

Data Properties

These properties appear in the "Data" category of the Properties Editor.

- **DataSourceID** (string) – Gets or sets the ID of the control from which the data-bound control retrieves its list of data items. This must be a Data Source that is compatible with Dynamic Data, including LinqDataSource, EntityDataSource, and DynamicObjectDataSource.

  See "Establishing a DataSource".

- **DataMember** (string) – Generally you don't use this since the DataSource control should supply the correct data with its own query capabilities.

- **DataKeyNames** (array of strings) – Gets or sets an array that contains the names of the primary key fields for the items displayed in a DynamicFormView control. See http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.formview.datakeynames.aspx

- **AutoScaffoldDataFields** (enum PeterBlum.DES.DynamicData.AutoScaffoldDataFields) – Determines if the **ItemsInPattern** collection is automatically populated with Data Fields using the Automatic Scaffolding rules.

  When in use, this adds DataFieldInPattern objects to the ItemsInPattern collection by using the Scaffolding rules. When ItemsInPattern already has objects, scaffolded objects are added at the end of the collection.

  This property is ignored when the **PartsToGenerate** property does not reflect a data oriented part, such as `ContainerOpeningTag` and `Buttons`. It is also ignored when the **DataField** property is assigned.

  The enumerated type AutoScaffoldDataFields has these values:

  - `True` – Use automatic scaffolding to populate **ItemsInPattern**.
  - `False` – Do not modify the **ItemsInPattern** collection.
  - `WhenEmpty` - The **ItemsInPattern** collection must be empty to use automatic scaffolding. This is the default.

- **ItemsInPattern** (PeterBlum.DES.DynamicData.ItemsInPattern) – Use this collection to identify the data fields that will be assigned to DynamicControls found in the Pattern Template. It can also specify formatting available through NamedParts of PartsPatternTemplates and TemplateNotInPattern objects.

  See "Defining a list of data fields".

  The ItemInPattern classes are:

  - The DataFieldInPattern Class
  - The DataFieldListInPattern Class
  - The ScaffoldInPattern Class
  - The NoDataFieldsInPattern Class
  - The TemplateNotInPattern Class

  If you need to programmatically find a specific object within **Fields**, there are two methods available on the DynamicFormView control:

  - For DataFieldInPattern objects, use `FindByDataField("datafieldname")`.

    See "GetAttributeAsInteger() method".

  - For all objects, assign a unique value in the ItemInPattern object's ID property when the object is first created or in the ASP.NET markup. Then use `FindByID("id")` and typecast to the expected class.

    See "FindByID() method".

- **PageIndex** (integer) – When **AllowPaging** is `true`, this determines the displayed page (which is an offset into a list of records starting with 0 as the first record). See http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.formview.pageindex.aspx.

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

PatternTemplate Properties

These properties appear in the "Pattern Template" category of the Properties Editor.

- **PatternTemplateName** (string) – The name of the Pattern Template file that generates the HTML for a single data field. The ".ascx" extension is optional.

  The Pattern Template must be a PartsPatternTemplate.

  It defaults to `"DetailsView"`, which creates a two column table that mimics the DetailsView control.

  If you assign this property to a file in a different folder from **[web application]/DynamicData/PatternTemplates**, set the URL to that folder in the **PatternTemplatesFolderUrl** property.

  For more, see "Choosing the PatternTemplate".

- **PatternTemplatesFolderUrl** (string) – The URL to the folder containing the Pattern Template file. If the file is in the **[web application]/DynamicData/PatternTemplates** folder, this property can be unassigned.

  It defaults to "".

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

Default NamedParts Properties

These properties appear in the "Default NamedParts" category of the Properties Editor.

Each Part in a PartsPattern Template can use a NamedPart instead of the part declared on the DynamicPartsPatterns control. The following chart indicates the properties that define a NamedPart by its name.

Part Type	Property name
Container	ContainerNamedPart
HeaderRow	HeaderRowNamedPart
DataRow	DataRowNamedPart
DataCell	DataCellNamedPart **DetailsView.ascx** has the following NamedParts: "TwoDataFields" and "SeveralDataFieldsIn1Row".
Buttons	ButtonsNamedPart **DetailsView.ascx** has NamedParts for LinkButtons and Standard buttons with these names: "LinkButtons", "Buttons", "LinkButtonsTogether", and "ButtonsTogether".
FooterRow	FooterRowNamedPart
NoDataToShow	NoDataToShowNamedPart

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

Appearance Properties

These properties appear in the "Appearance" category of the Properties Editor.

- **NamedStyles** (PeterBlum.DES.DynamicData.NamedStyles) – A list of NamedStyle objects. These are passed to all Pattern Templates generated within this DynamicFormView control.

  Named Styles allow the user to deliver formatting rules into your Pattern Template. You declare various HTML tags that they can format with a DynamicStylesTag control, or by calling the `GetTagAttributes()` method. These take a name which is used to lookup a NamedStyle object from this list. (You don't have to search this list yourself.)

  For more details, see "Using Named Styles to adjust the formatting".

- **AutoGenerateButtons** (enum PeterBlum.DES.DynamicData.AutoGenerateButtons) – Determines if and where Buttons will appear on each row. The appearance for buttons is determined by the Pattern Template in **ButtonsNamedPart**.

  The enumerated type `PeterBlum.DES.DynamicData.AutoGenerateButtons` has these values:

  - `First` - first column.
  - `Last` - last column. This is the default.
  - `FirstLast` - both first and last.
  - `None` - do not auto generate. Often the user declares an ButtonsInPattern object for the desired template in the **Fields** property.

```
<desDD:DynamicFormView id="DynamicFormView1" runat="server"
   DataSourceID="RecordsDataSource" AutoScaffoldDataFields="false"
   AutoGenerateButtons="None" >
   <ItemsInPattern>
      <desDD:ScaffoldInPattern />
      <desDD:ButtonsInPattern NamedPart="Buttons" />
   </ItemsInPattern>
</desDD:DynamicFormView>
```

- **ShowHeaderRow** (Boolean) – Determines if the Header Row is added. The PartsPatternTemplate defines the HTML to output in its `<HeaderRow>` Part. See "Header Row and Column Titles".

  When `true`, the Header Row is shown.

  When `false`, the Header Row is not shown.

  It defaults to `false`.

- **HeaderRules** (enum PeterBlum.DES.DynamicData.DynamicListViewHeaderRules) – Determines whether to use the automatically generated header row or **HeaderTemplate**. It can offer both and determine the order they are shown.

  The header row is only used when **ShowHeaderRow** is `true`.

  The enumerated type `PeterBlum.DES.DynamicData.DynamicListViewHeaderRules` has these values:

  - `Normal` - If **HeaderTemplate** is defined, use it. Otherwise use the automatic generated header row. This is the default.
  - `AutoOnly` - Only use the automatic generated header row.
  - `TemplateOnly` - Only use the **HeaderTemplate**. If not defined, use nothing.
  - `AutoThenTemplate` - First output the automatic generated header row. Then output the **HeaderTemplate**.
  - `TemplateThenAuto` - First output the **HeaderTemplate**. Then output the automatic generated header row.

- **HeaderTemplate** (ITemplate) – The controls and HTML that form a header row. It can be used together with or in place of the automatically generated header row based on the **HeaderRules** property.

  Use this to customize the header beyond what is available within the Pattern Template that format the column titles.

  Often used to create a table row on a list view.

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

The header row is only used when **ShowHeaderRow** is `true`.

For details, see "Header Row and Column Titles".

- **HeaderText** (string) – Used when **HeaderTemplate** is unassigned to create text prior to the record shown. See http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.formview.headertext.aspx.
- **ShowFooterRow** (Boolean) – Determines if the Footer Row is added. The PartsPatternTemplate defines the HTML to output in its `<FooterRow>` Part. See "Footer Row".

  When `true`, the Footer Row is shown.

  When `false`, the Footer Row is not shown.

  It defaults to `false`.
- **FooterRules** (enum PeterBlum.DES.DynamicData.DynamicListViewFooterRules) – Determines whether to use the automatically generated Footer Row or **FooterTemplate**. It can offer both and determine the order they are shown.

  The Footer row is only used when **ShowFooterRow** is `true`.

  The enumerated type `PeterBlum.DES.DynamicData.DynamicListViewFooterRules` has these values:

  - `Normal` - If **FooterTemplate** is defined, use it. Otherwise use the automatic generated Footer row. This is the default.
  - `AutoOnly` - Only use the automatic generated Footer row.
  - `TemplateOnly` - Only use the **FooterTemplate**. If not defined, use nothing.
  - `AutoThenTemplate` - First output the automatic generated Footer row. Then output the **FooterTemplate**.
  - `TemplateThenAuto` - First output the **FooterTemplate**. Then output the automatic generated Footer row.
- **FooterTemplate** (ITemplate) – The controls and HTML that form a footer row. It can be used together with or in place of the automatically generated header row based on the **FooterRules** property.

  The footer row is only used when **ShowFooterRow** is `true`.

  For details, see "Footer Row".
- **FooterText** (string) – Used when **FooterTemplate** is unassigned to create text prior to the record shown. See http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.formview.Footertext.aspx.
- **EmptyDataText** (string) – Gets or sets the text to display in the empty data row rendered when this control is bound to a data source that does not contain any records. It defaults to "".

  See http://msdn.microsoft.com/en-us/library/system.web.ui.webcontrols.formview_properties.aspx.

  When used, it replaces the content generated by the `<NoDataToShow>` Part of the PartsPatternTemplate.

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

Template Properties

- **ItemTemplate** (ITemplate) – A body row used for read-only mode. When used, it replaces rows that are automatically generated using the `<DataCell>` Part within the PartsPatternTemplate. You must create all of the HTML and controls to do the work that was automatically generated. See "Using FormView's Templates instead of automatic generation".
- **EditItemTemplate** (ITemplate) – A body row used for Edit mode. When used, it replaces rows that are automatically generated using the `<DataCell>` Part within the PartsPatternTemplate. You must create all of the HTML and controls to do the work that was automatically generated. See "Using FormView's Templates instead of automatic generation".
- **InsertItemTemplate** (ITemplate) – A body row used for Insert mode. When used, it replaces rows that are automatically generated using the `<DataCell>` Part within the PartsPatternTemplate. You must create all of the HTML and controls to do the work that was automatically generated. See "Using FormView's Templates instead of automatic generation".

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

Scaffold Caching Properties

These properties appear in the "Scaffold Caching" category of the Properties Editor.

Automatic Scaffolding can cache the results of collecting a list of DataFieldInPattern objects to avoid generating the same list multiple times. These properties determine if scaffolding is used and how long data is preserved in the cache.

- **EnableScaffoldCache** (enum PeterBlum.DES.DynamicData.CacheLocation) – Determines if caching is used and its location when on.

  The user can turn off caching on an individual request simply by setting this to `None`.

  Cached data is assigned to a group with a named defined in **CacheCategoryName**. Separate cached data is kept by the MetaTable.Name requesting the data and by the data entry mode. The actual cache name combines all of these elements.

  The enumerated type CacheLocation has these values:

  - None - Cache is not used. Scaffolding is requested on each DynamicPattern control.
  - Page - Cache lasts for the page request. This allows many DynamicPatterns that work together to create a single data bound control to share a list of DataFieldInPattern objects. This is the default.
  - `User` - Cache per user using the Session class.
  - Global - Cache using the HttpCache class with a priority and timeout.

- **CacheCategoryName** (string) – Defines a category for this cache. Categories let you have multiple caches, for different situations.

  You might have a different dataset per User's role. This could be the role name.

  It defaults to "".

- **CacheExpirationSeconds** (integer) – Determines that entries cached in the cache expire after this many seconds.

  When in the Global cache, ASP.NET will automatically delete them when no longer needed. When in the User cache, they are only abandoned if requested and over this time limit. Not used when using the Page cache.

  It defaults to 60 (seconds).

- **CacheSlidingExpiration** (Boolean) – When using the Global or User cache, this determines if the expiration time (**CacheExpirationSeconds**) resets each time the item is retrieved from the cache.

  When `true`, expiration time resets each time the item is retrieved.

  When `false`, the expiration time is from the time initially added to the cache.

  It defaults to `false`.

- **GlobalCachePriority** (enum System.Web.Caching.CacheItemPriority) – When using the Global cache, this is the priority to preserve the data when the cache needs more memory.

  It defaults to `CacheItemPriority.Normal`.

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

Behavior Properties

- **Visible** (Boolean) – When `false`, this control is not used. It defaults to `true`.
- **AllowPaging** (Boolean) – Activates the built-in Paging feature of FormView. However, its not recommended. Use DES's DynamicControlPanel instead to provide a richer paging system. See "Offering Paging".
- **EnableModelValidation** (Boolean) – Gets or sets a value that indicates whether a validator control will handle exceptions that occur during insert or update operations. While it defaults to `false`, by using it with Dynamic Data, the DynamicFormViewAdapter will set it to `true` during a postback or callback. Generally you do not modify this property.
- **DefaultMode** (enum System.Web.UI.WebControls.DataBoundControlMode) - Gets or sets the data-entry mode to which the DynamicFormView control returns after an update, insert, or cancel operation.
- **AutoLoadPatternTemplateStyleSheetFile** (Boolean) – Pattern Templates often use the style sheet classes defined in the file: **~/DynamicData/PatternTemplateNamedStyles.css**. When this property is `true`, that file is automatically added to the page without you establishing a Link tag.

  When used, it merges the style sheet file with other DES files. It also compresses the file, removing comments and much whitespace.

  When not used, you can still load it merged and compressed by calling this in Page_Load().

  ```
  PeterBlum.DES.Globals.Page.StyleSheetManager.Include(
      PeterBlum.DES.StyleSheetFiles.PatternTemplateNamedStyles)
  ```

  The path to this file can be changed in two ways:

  In `Application_Start()`, call

  ```
  PeterBlum.DES.StyleSheetManager.SetDefaultUrl(
      PeterBlum.DES.StyleSheetFiles.PatternTemplateNamedStyles, "URL")
  ```

  In the `<appSettings>` section of **web.config**, add

  ```
  <add key="DES_StyleSheetPatternTemplateNamedStyles" value="URL" />
  ```

  It autoloads when `true`.

  It defaults to `true`.

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

Using the PatternTemplates shipped with DES Dynamic Data

Click on any of these topics to jump to them:

- Using DetailsView.ascx PatternTemplate
- Using GridView.ascx PatternTemplate
- Using ListOfDetailViews.ascx PatternTemplate
- Using TwoColumnTableRow.ascx PatternTemplate
- Using TwoStacked.ascx PatternTemplate
- Using DetailsViewButtons.ascx and DetailsViewLinkButtons.ascx PatternTemplates
- Using ButtonsForList.ascx and LinkButtonsForList.ascx PatternTemplates
- Using ButtonsForSingle.ascx and LinkButtonsForSingle.ascx PatternTemplates

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

Using DetailsView.ascx PatternTemplate

DetailsView.ascx is a Pattern Template subclassed from `PeterBlum.DES.DynamicData.PartsPatternTemplate`. It mimics the appearance of the DetailsView web control to show a single record. It is the default Pattern Template for the DynamicFormView control.

ID	1
Product Name	Chai!
Price	$31.50
Reorder Level	3 and Units In Stock : 11 gallons
Discontinued	Units In Stock: 11 gallons
	Edit New

The default setup will give you everything automatically. You only need to define the list of data fields.

Each data field is output with this format.

```
<tr>
   <td>[DynamicLabel]</td>
   <td>[DynamicControl]</td>
</tr>
```

There can be separate definitions for ReadOnly, Edit, and Insert modes to allow further customization by adding `<EditTemplate>` and `<InsertTemplate>` nodes to the `<DataCell>` Part in the Pattern Template file.

Click on any of these topics to jump to them:

- Example: DynamicPattern in FormView
- Example: DynamicPattern with quick setup
- Example: DynamicFormView
- NamedStyles on the DetailsView PatternTemplate
- NamedParts on the DetailsView PatternTemplate
- Properties on the DetailsView PatternTemplate

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

Example: DynamicPattern in FormView

Define DynamicPattern controls with the appropriate PartsToGenerate and Mode properties in the Templates as shown here.

```
<asp:FormView ID="FormView1" runat="server" DataSourceID="FormDataSource" >
<HeaderTemplate>
  <desDD:DynamicPattern id="DPOpeningTag" runat="server"
    PatternTemplateName="DetailsView" PartsToGenerate="ContainerOpeningTag" />
</HeaderTemplate>
<ItemTemplate>
  <desDD:DynamicPattern id="DPReadOnly" runat="server"
    PatternTemplateName="DetailsView" PartsToGenerate="DataRow"
    AutoGenerateButtons="Last" >
  </desDD:DynamicPattern>
</ItemTemplate>
<EditItemTemplate>
  <desDD:DynamicPattern id="DPEdit" runat="server"
    PatternTemplateName="DetailsView" PartsToGenerate="DataRow" Mode="Edit"
    AutoGenerateButtons="Last" >
  </desDD:DynamicPattern>
</EditItemTemplate>
<InsertItemTemplate>
  <desDD:DynamicPattern id="DPInsert" runat="server"
    PatternTemplateName="DetailsView" PartsToGenerate="DataRow" Mode="Insert"
    AutoGenerateButtons="Last">
  </desDD:DynamicPattern>
</InsertItemTemplate>
<FooterTemplate>
  <desDD:DynamicPattern id="DPClosingTag" runat="server"
    PatternTemplateName="DetailsView" PartsToGenerate="ContainerClosingTag" />
</FooterTemplate>
</asp:FormView>
```

Example: DynamicPattern with quick setup

Several **PartsToGenerate** values put together all of the other parts: container opening tag, header row, detail row, footer row, and container closing tag. They are DetailView, DetailViewNoHeader, DetailViewNoFooter, and DetailViewNoHeaderFooter. Put this in the FormView's `<ItemTemplate>`, `<EditItemTemplate>`, and `<InsertItemTemplate>` tags

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
  PatternTemplateName="DetailsView" PartsToGenerate="DetailViewNoHeaderFooter"
  Mode="ReadOnly, Edit, or Insert" AutoGenerateButtons="Last">
</desDD:DynamicPattern>
```

Example: DynamicFormView

The DynamicFormView uses **DetailsView.ascx** by default. It also establishes a default for **AutoGenerateButtons** as `Last`.

```
<desDD:DynamicFormView ID="DynamicFormView1" runat="server"
  DataSourceID="DataSourceID" >
</desDD:DynamicFormView>
```

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

NamedStyles on the DetailsView PatternTemplate

Add `PeterBlum.DES.DynamicData.NamedStyle` objects to the **NamedStyles** property on the DynamicPattern control to declare any of these style names. The style sheet classes are declared in the **[web app]/DynamicData/Content/PatternTemplateNamedStyles.css** file.

NamedStyle	Usage	Default style sheet class
"Container"	Style for the container tag around the entire data bound control.	"DetailsViewContainer"
"LabelColumn"	Style for the first column (`<td>` tag), containing the label.	"DetailsViewLabelColumn"
"LabelControl"	Style for the DynamicLabel control in the first column.	"DetailsViewLabelControl"
"DataColumn"	Style for the second column (`<td>` tag), containing the data.	"DetailsViewDataColumn"
"DataControl"	Style for the DynamicControl in the second column.	"DetailsViewDataControl"
"DataFieldRow"	Style for the `<tr>` tag enclosing one data field	"DetailsViewDataFieldRow"
"AltDataFieldRow"	Style for the `<tr>` tag enclosing one data field with an alternative style.	"DetailsViewAltDataFieldRow"
"HeaderRow"	Style for the optional column headers.	"DetailsViewHeaderRow"
"FooterRow"	Style for the optional column footers.	"DetailsViewFooterRow"
"LabelColumn2"	Style for the `<div>` tag containing the DynamicLabel in the"SeveralDataFieldsIn1Row" NamedPart. If not defined, it uses "LabelColumn". Requires setting the **UseSDFLabelContainers** property to `true` to establish the `<div>` tag.	"DetailsViewLabelColumn2"
"DataColumn2"	Style for the `<div>` tag containing the DynamicControl in the"SeveralDataFieldsIn1Row" NamedPart. If not defined, it uses "DataColumn".	"DetailsViewDataColumn2"
"HeaderLabelColumn"	Style for the first column (`<td>` tag) in the HeaderTemplate. If not defined, it uses "LabelColumn".	"DetailsViewHeaderLabelColumn"
"HeaderDataColumn"	Style for the second column (`<td>` tag) in the HeaderTemplate. If not defined, it uses "DataColumn".	"DetailsViewHeaderDataColumn"
"FooterLabelColumn"	Style for the first column (`<td>` tag) in the FooterTemplate. If not defined, it uses "LabelColumn".	"DetailsViewFooterLabelColumn"
"FooterDataColumn"	Style for the second column (`<td>` tag) in the FooterTemplate. If not defined, it uses "DataColumn".	"DetailsViewFooterDataColumn"
"ButtonsContainer"	Style around the buttons.	"DetailsViewButtonsContainer"
"NoDataToShow"	Style for the content displayed when there is no data to show.	"DetailsViewNoDataToShow"

Example

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
   PatternTemplateName="DetailsView" PartsToGenerate="DataRow" Mode="Edit" >
   <NamedStyles>
      <desDD:NamedStyle Name="LabelColumn" CssClass="MyClass" />
      <desDD:NamedStyle Name="DataColumn" Style="width:200px;" />
   </NamedStyles>
</desDD:DynamicPattern>
```

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

NamedParts on the DetailsView PatternTemplate

There are additional Templates in this file under the `<NamedParts>` nodes of various Parts. When used, you get an alternative user interface that can show just about anything.

NamedParts can be used in the following ways:

- A default NamedPart name can be specified in properties on the DynamicPattern and DynamicFormView controls. Each property has this format: ***PartType*NamedPart**. For example, to use the "Buttons" named part, use ButtonsNamedPart="Buttons".
- Assign the name to the **NamedPart** property on an ItemInPattern object (DataFieldInPattern, ButtonsInPattern, DataFieldsListInPattern, etc.)

Here are the predefined NamedParts in the DetailsView Pattern Template. Feel free to add your own or customize these:

Name/PartType	Description
"Buttons" in `<Buttons>`	Uses the **DetailsViewButtons.ascx** Pattern Template file, which provides a table row with the appropriate buttons for use in Read Only, Edit, and Insert modes. Standard Buttons are used.
"LinkButtons" in `<Buttons>`	Uses the **DetailsViewLinkButtons.ascx** Pattern Template file, which provides a table row with the appropriate buttons for use in Read Only, Edit, and Insert modes. LinkButtons are used.
"ButtonsTogether" in `<Buttons>`	Uses the **ButtonsForSingle.ascx** Pattern Template file, which provides a space delimited list of buttons for use in Read Only, Edit, and Insert modes. Standard Buttons are used.
"LinkButtonsTogether" in `<Buttons>`	Uses the **LinkButtonsForSingle.ascx** Pattern Template file, which provides a space delimited list of buttons for use in Read Only, Edit, and Insert modes. LinkButtons are used.

"AltStyle" in `<DataCell>`	Specifies a different style sheet and NamedStyle ("AlternativeDataFieldRow") for the `<tr>` tag around the row. It uses the standard row's content.
	It is a good way to differentiate an important field like, like the product name, from the rest.
	`<ItemsInPattern>`
	`  <desDD:DataFieldInPattern DataField="ProductName"`
	`  NamedPart="AltStyle" />`
	`  <desDD:DataFieldInPattern DataField="Description" />`
	`</ItemsInPattern>`

"TwoDataFields" in `<DataCell>`	Shows two data fields and their labels. It has this format:
	`<td>Label1</td><td>Data field1[`**`SDF_HTMLBeforeLabel`**`]`
	`<div>Labe12</div>[`**`SDF_HTMLLabelDataSeparator`**`]`
	`<div>Data field2</div>[`**`SDF_HTMLAfterData`**`]</td>`
	The bold elements are property names defined on DetailsView.ascx. See "Properties on the DetailsView PatternTemplate"
	Here's how to use this NamedPart:
	`<desDD:DataFieldListInPattern NamedPart="TwoDataFields">`
	`  <ItemsInPattern>`
	`  <desDD:DataFieldInPattern DataField="Columnname1" />`
	`  <desDD:DataFieldInPattern DataField="Columnname2" />`
	`  </ItemsInPattern>`
	`</desDD:DataFieldListInPattern>`

"SeveralDataFields-In1Row" in `<DataCell>`	Creates a table row with two table cells. It displays one or more Data Fields, depending on the number of DataFieldInPattern objects added to the DataFieldInListPattern object. It is similar to the "TwoDataFields" NamedPart, using the same basic HTML output where all data fields are shown side-by-side in the second table column. There are several properties declared on this Pattern Template to modify the behavior of "SeveralDataFieldsIn1Row". See "Properties on the DetailsView PatternTemplate"

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

Properties on the DetailsView PatternTemplate

To set these properties, see "Passing properties to be used by the PatternTemplate's own code".

- **HeaderLabelColumnHTML** (string) – When using the HeaderTemplate, it shows this text in the first column. It defaults to "`Name`".
- **HeaderDataColumnHTML** (string) – When using the HeaderTemplate, it shows this text in the second column. It defaults to "`Value`".
- **FooterLabelColumnHTML** (string) – When using the FooterTemplate, it shows this text in the first column. It defaults to "` `".
- **FooterDataColumnHTML** (string) – When using the FooterTemplate, it shows this text in the second column. It defaults to "` `".
- **SDF_HTMLBeforeLabel** (string) – Used with the "`SeveralDataFieldsIn1Row`" and "`TwoDataFields`" NamedParts. From the second column on, this HTML is inserted before the label providing separation from the previous column's UI.

  It defaults to "` `".
- **SDF_HTMLLabelDataSeparator** (string) – Used with the "`SeveralDataFieldsIn1Row`" and "`TwoDataFields`" NamedParts. From the second column on, this HTML is inserted between the label and data field.

  It defaults to ":` `".
- **SDF_HTMLAfterLabel** (string) – Used with the "`SeveralDataFieldsIn1Row`" and "`TwoDataFields`" NamedParts. From the second column on, this HTML is inserted after the controls showing data providing separation from the next column's UI.

  It defaults to "".
- **UseSDFLabelContainers** (bool) – Used with the "`SeveralDataFieldsIn1Row`" NamedPart. Establish a `<div>` tag around labels. When used, you can setup the NameStyle = "DetailsViewLabelColumn2" to adjust the div tag's layout (such as its width).

  It defaults to `false`.
- **UseSDFDataContainers** (bool) – Used with the "`SeveralDataFieldsIn1Row`" NamedPart. Establish a `<div>` tag around data controls. When used, you can setup the NameStyle = "DetailsViewDataColumn2" to adjust the `<div>` tag's layout (such as its width).

  It defaults to `false`.

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

Using GridView.ascx PatternTemplate

GridView.ascx is a Pattern Template subclassed from `PeterBlum.DES.DynamicData.PartsPatternTemplate`. It mimics the appearance of the GridView web control. It is the default Pattern Template for the DynamicListView control.

	Product Name	Unit Price	Quantity Per Unit	Reorder Level	Units In Stock	Discontinued
Edit	Chai!	$31.50	10 boxes x 20 bags	3	11 gallons	●
Edit	Chang!	$18.00	24 - 12 oz bottles	10	17 gallons	
Edit	Aniseed Syrup	$10.00	12 - 550 ml bottles	5	15 gallons	
Edit	Chef Anton's Cajun Seasoning	$23.00	48 - 6 oz jars	4	53 gallons	
Edit	Chef Anton's Gumbo Mix	$21.35	36 boxes	1	0 gallons	●

The default setup will give you everything automatically You only need to define the list of data fields.

Each data field is output with this format.

```
<table>
   <tr>List of: <td>[DynamicColumnTitle]</td></tr>          Header row
   List of: <tr> List of: <td>[DynamicControl]</td></tr>    Data rows
   <tr>List of: <td>[DynamicLabel]</td></tr>                Footer rows
</table>
```

There can be separate definitions for ReadOnly, Edit, and Insert modes to allow further customization by adding <EditTemplate> and <InsertTemplate> nodes to the <DataCell> Part in the Pattern Template file.

Click on any of these topics to jump to them:

- Example: DynamicPattern in ListView
- Example: DynamicListView
- NamedStyles on the GridView PatternTemplate
- NamedParts on the GridView PatternTemplate

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

Example: DynamicPattern in ListView

Define DynamicPattern controls with the appropriate **PartsToGenerate** and **Mode** properties in the Templates as shown here.

```
<asp:ListView ID="ListView1" runat="server" DataSourceID="ListDataSource" >
<LayoutTemplate>
   <desDD:DynamicPattern ID="DPOpeningTag" runat="server"
      PatternTemplateName="GridView" PartsToGenerate="ContainerOpeningTag" />

   <desDD:DynamicPattern id="DPHeaderRow" runat="server" AutoGenerateButtons="Last"
      PatternTemplateName="GridView" PartsToGenerate="HeaderRow" >
   </desDD:DynamicPattern>

   <asp:PlaceHolder runat="server" id="itemPlaceholder" />

   <desDD:DynamicPattern id="DPFooterRow" runat="server" AutoGenerateButtons="Last"
      PatternTemplateName="GridView" PartsToGenerate="FooterRow" >

   <desDD:DynamicPattern ID="DPClosingTag" runat="server"
      PatternTemplateName="GridView" PartsToGenerate="ContainerClosingTag" />
</LayoutTemplate>

<ItemTemplate>
   <desDD:DynamicPattern id="DPReadOnly" runat="server" AutoGenerateButtons="Last"
      PatternTemplateName="GridView" PartsToGenerate="DataRow" Mode="ReadOnly" >
   </desDD:DynamicPattern>
</ItemTemplate>

<EditItemTemplate>
   <desDD:DynamicPattern id="DPEdit" runat="server" AutoGenerateButtons="Last"
      PatternTemplateName="GridView" PartsToGenerate="DataRow" Mode="Edit"  >
   </desDD:DynamicPattern>
</EditItemTemplate>

<InsertItemTemplate>
   <desDD:DynamicPattern id="DPInsert" runat="server" AutoGenerateButtons="Last"
      PatternTemplateName="GridView" PartsToGenerate="DataRow" Mode="Insert" >
   </desDD:DynamicPattern>
</InsertItemTemplate>

<SelectedItemTemplate>
   <desDD:DynamicPattern id="DPSelect" runat="server" AutoGenerateButtons="Last"
      PatternTemplateName="GridView" PartsToGenerate="DataRow" Mode="ReadOnly"
      DataRowAppearance="Selected" ShowSelectButton="false" >
   </desDD:DynamicPattern>
</SelectedItemTemplate>

<AlternatingItemTemplate>
   <desDD:DynamicPattern id="DPAlt" runat="server" AutoGenerateButtons="Last"
      PatternTemplateName="GridView" PartsToGenerate="DataRow" Mode="ReadOnly"
      DataRowAppearance="Alternative" >
   </desDD:DynamicPattern>
</AlternatingItemTemplate>

<EmptyDataTemplate>
   <desDD:DynamicPattern id="DPReadOnly" runat="server"
      PatternTemplateName="GridView" PartsToGenerate="NoDataToShow" >
   </desDD:DynamicPattern>
</EmptyDataTemplate>

</asp:ListView>
```

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

Example: DynamicListView

The DynamicListView uses **GridView.ascx** by default. It also establishes a default for **AutoGenerateButtons** as `First`.

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
   DataSourceID="DataSourceID" >
</desDD:DynamicListView>
```

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

NamedStyles on the GridView PatternTemplate

Add `PeterBlum.DES.DynamicData.NamedStyle` objects to the **NamedStyles** property on the DynamicPattern control to declare any of these style names. The style sheet classes are declared in the **[web app]/DynamicData/Content/PatternTemplateNamedStyles.css** file.

NamedStyle	Usage	Default style sheet class
“Container”	Style for the container tag around the entire data bound control.	"GridViewContainer"
“HeaderRow”	Style for the `<tr>` tag on the header row. If not defined, it falls back to "DataRow".	“GridViewHeaderRow
“HeaderCell”	Style for the table header cells (`<th>` tag)	“GridViewHeaderCell”
“HeaderControl”	Style for label controls in the header row.	“GridViewHeaderControl”
“DataRow”	Style for the `<tr>` tag.	“GridViewDataRow”
“DataCell”	Style for table cells (`<td>` tag) containing the data.	“GridViewDataCell”
“DataControl”	Style for any DynamicControl in a DataRow.	“GridViewDataControl”
“DataLabel”	Style for any DynamicLabel in a DataRow.	“GridViewDataLabel”
“SelectedRow”	Style for the `<tr>` tag on a DataRow that is marked “selected”.	“GridViewSelectedRow”
“AlternativeRow”	Style for the `<tr>` tag on alternative Data Rows.	“GridViewAlternativeRow”
“ButtonsCell”	Style for columns (`<td>` tag) that contain Buttons. If not defined, it uses "DataColumn".	“GridViewButtonsCell”
“DataRowSeparator”	Style for the container tag around a DataRow Separator.	“GridViewDataRowSeparator”
“FooterRow”	Style for the `<tr>` tag on the Footer row. If not defined, it falls back to "DataRow".	“GridViewFooterRow
“FooterCell”	Style for the table Footer cells (`<th>` tag)	“GridViewFooterCell”
“FooterControl”	Style for label controls in the Footer row.	“GridViewFooterControl”
“NoDataToShow”	Style for the NoDataToShow part	“GridViewNoDataToShow”

Example

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
   PatternTemplateName="GridView" >

   <NamedStyles>
      <desDD:NamedStyle Name="HeaderCell" CssClass="MyClass" />
      <desDD:NamedStyle Name="DataCell" Style="width:200px;" />
   </NamedStyles>

</desDD:DynamicPattern>
```

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

NamedParts on the GridView PatternTemplate

There are additional Templates in this file under the `<NamedParts>` nodes of various Parts. When used, you get an alternative user interface that can show just about anything.

NamedParts can be used in the following ways:

- A default NamedPart name can be specified in properties on the DynamicPattern and DynamicListView controls. Each property has this format: ***PartType*NamedPart**. For example, to use the "Buttons" named part, use ButtonsNamedPart="Buttons".
- Assign the name to the **NamedPart** property on an ItemInPattern object (DataFieldInPattern, ButtonsInPattern, DataFieldsListInPattern, etc.)

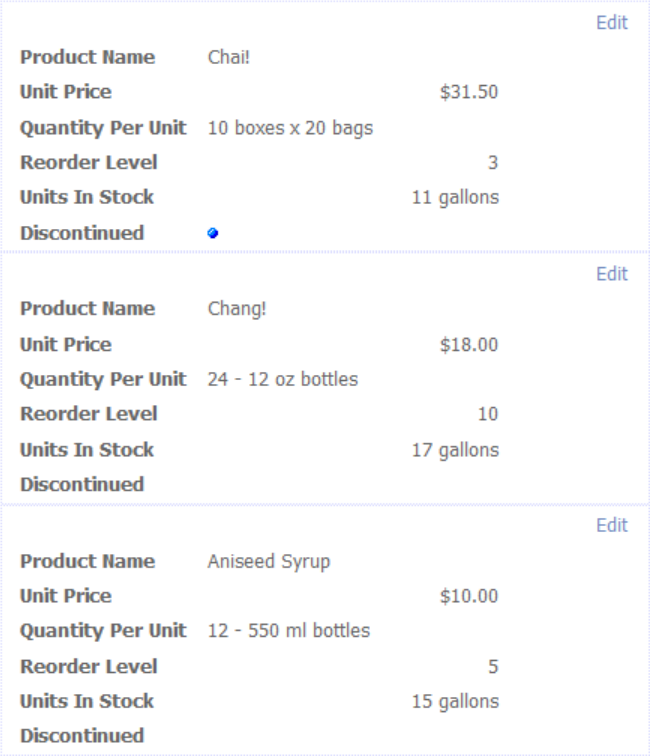
Here are the predefined NamedParts in the GridView Pattern Template. Feel free to add your own or customize these:

Name/PartType	Description
"Buttons" in `<Buttons>`	Uses the **ButtonsForList.ascx** Pattern Template file, which provides a table row with the appropriate buttons for use in Read Only, Edit, and Insert modes. Standard Buttons are used.
"LinkButtons" in `<Buttons>`	Uses the **LinkButtonsForList.ascx** Pattern Template file, which provides a table row with the appropriate buttons for use in Read Only, Edit, and Insert modes. LinkButtons are used.
"SelectButton" in `<Buttons>`	Shows only single button for the Select command. Standard Buttons are used. Choose this when the DynamicControlPanel contains a DynamicButtonsField to offer the rest of the buttons.
"LinkSelectButton" in `<Buttons>`	Same as "SelectButton" but using LinkButtons.
"NoSort" in `<HeaderCell>`	Provides a column title without the sorting user interface that is offered by default.
"TwoStacked" in `<HeaderCell>`	Provides two column titles, separated by a ` ` tag. Here's how to use this NamedPart: `<desDD:DataFieldListInPattern NamedPart="TwoStacked">` `  <ItemsInPattern>` `  <desDD:DataFieldInPattern DataField="Columnname1" />` `  <desDD:DataFieldInPattern DataField="Columnname2" />` `  </ItemsInPattern>` `</desDD:DataFieldListInPattern>`
"TwoStacked" in `<DataCell>`	Shows two data fields separated by a ` ` tag. It has this format: `Data field1 ` `Data field2` The bold elements are property names defined on DetailsView.ascx. See "Properties on the DetailsView PatternTemplate" Here's how to use this NamedPart: `<desDD:DataFieldListInPattern NamedPart="TwoDataFields">` `  <ItemsInPattern>` `  <desDD:DataFieldInPattern DataField="Columnname1" />` `  <desDD:DataFieldInPattern DataField="Columnname2" />` `  </ItemsInPattern>` `</desDD:DataFieldListInPattern>`
"ShowLabel" in `<DataCell>`	A single DynamicLabel control. It can either show the name of the data field or you can use the LabelInPattern control to define text in the LabelText property. `<desDD:LabelInPattern LabelText="<img src='picture.jpg' />" />`

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

Using ListOfDetailViews.ascx PatternTemplate

ListOfDetailViews.ascx is a Pattern Template subclassed from `PeterBlum.DES.DynamicData.PartsPatternTemplate`. Use it to create list-style interfaces where each item is a two column table that mimics the appearance of the DetailsView control. It supports the Grouping feature of ListView and DynamicListView controls.



The default setup will give you a complete row automatically You only need to define the list of data fields and buttons.

There can be separate definitions for ReadOnly, Edit, and Insert modes to allow further customization by adding `<EditTemplate>` and `<InsertTemplate>` nodes to the `<DataCell>` Part in the Pattern Template file.

Click on any of these topics to jump to them:

- Example: DynamicPattern in ListView
- Example: ListView in using Groups
- Example: DynamicListView
- Example: DynamicListView using Groups
- NamedStyles on the ListOfDetailViews PatternTemplate
- NamedParts on the ListOfDetailViews PatternTemplate

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

Example: DynamicPattern in ListView

Define DynamicPattern controls with the appropriate **PartsToGenerate** and **Mode** properties in the Templates as shown here.

```
<asp:ListView ID="ListView1" runat="server" DataSourceID="ListDataSource" >
<LayoutTemplate>
   <desDD:DynamicPattern ID="DPOpeningTag" runat="server"
      PatternTemplateName="ListOfDetailViews" PartsToGenerate="ContainerOpeningTag" />

   <desDD:DynamicPattern id="DPHeaderRow" runat="server" AutoGenerateButtons="Last"
      PatternTemplateName="ListOfDetailViews" PartsToGenerate="HeaderRow" >
   </desDD:DynamicPattern>

   <asp:PlaceHolder runat="server" id="itemPlaceholder" />

   <desDD:DynamicPattern id="DPFooterRow" runat="server" AutoGenerateButtons="Last"
      PatternTemplateName="ListOfDetailViews" PartsToGenerate="FooterRow" >

   <desDD:DynamicPattern ID="DPClosingTag" runat="server"
      PatternTemplateName="ListOfDetailViews" PartsToGenerate="ContainerClosingTag" />
</LayoutTemplate>

<ItemTemplate>
   <desDD:DynamicPattern id="DPReadOnly" runat="server" AutoGenerateButtons="Last"
      PatternTemplateName="ListOfDetailViews" PartsToGenerate="DataRow"
      Mode="ReadOnly" >
   </desDD:DynamicPattern>
</ItemTemplate>

<EditItemTemplate>
   <desDD:DynamicPattern id="DPEdit" runat="server" AutoGenerateButtons="Last"
      PatternTemplateName="ListOfDetailViews" PartsToGenerate="DataRow" Mode="Edit"  >
   </desDD:DynamicPattern>
</EditItemTemplate>

<InsertItemTemplate>
   <desDD:DynamicPattern id="DPInsert" runat="server" AutoGenerateButtons="Last"
      PatternTemplateName="ListOfDetailViews" PartsToGenerate="DataRow" Mode="Insert" >
   </desDD:DynamicPattern>
</InsertItemTemplate>

<SelectedItemTemplate>
   <desDD:DynamicPattern id="DPSelect" runat="server" AutoGenerateButtons="Last"
      PatternTemplateName="ListOfDetailViews" PartsToGenerate="DataRow"
      Mode="ReadOnly"
      DataRowAppearance="Selected" ShowSelectButton="false" >
   </desDD:DynamicPattern>
</SelectedItemTemplate>

<AlternatingItemTemplate>
   <desDD:DynamicPattern id="DPAlt" runat="server" AutoGenerateButtons="Last"
      PatternTemplateName="ListOfDetailViews" PartsToGenerate="DataRow"
      Mode="ReadOnly"
      DataRowAppearance="Alternative" >
   </desDD:DynamicPattern>
</AlternatingItemTemplate>

<EmptyDataTemplate>
   <desDD:DynamicPattern id="DPReadOnly" runat="server"
      PatternTemplateName="ListOfDetailViews" PartsToGenerate="NoDataToShow" >
   </desDD:DynamicPattern>
</EmptyDataTemplate>

</asp:ListView>
```

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

Example: ListView in using Groups

This is an extension of the previous example, where LayoutTemplate now handles groups; GroupTemplate handles items; and GroupSeparatorTemplate and ItemSeparatorTemplate are setup.

Define DynamicPattern controls with the appropriate **PartsToGenerate** and **Mode** properties in the Templates as shown here.

```
<asp:ListView ID="ListView1" runat="server" DataSourceID="ListDataSource" >
<LayoutTemplate>
  <desDD:DynamicPattern ID="DPGroupOpeningTag" runat="server"
    PatternTemplateName="ListOfDetailViews"
    PartsToGenerate="GroupContainerOpeningTag" />

  <desDD:DynamicPattern id="DPGroupHeader" runat="server"
    PatternTemplateName="ListOfDetailViews" PartsToGenerate="GroupHeader" >
  </desDD:DynamicPattern>

  <asp:PlaceHolder runat="server" id="groupPlaceholder" />

  <desDD:DynamicPattern id="DPGroupFooter" runat="server"
    PatternTemplateName="ListOfDetailViews" PartsToGenerate="GroupFooter" >

  <desDD:DynamicPattern ID="DPGroupClosingTag" runat="server"
    PatternTemplateName="ListOfDetailViews"
    PartsToGenerate="GroupContainerClosingTag" />
</LayoutTemplate>

<GroupTemplate>
  <desDD:DynamicPattern ID="DPOpeningTag" runat="server"
    PatternTemplateName="ListOfDetailViews" PartsToGenerate="ContainerOpeningTag" />

  <desDD:DynamicPattern id="DPHeaderRow" runat="server" AutoGenerateButtons="Last"
    PatternTemplateName="ListOfDetailViews" PartsToGenerate="HeaderRow" >
  </desDD:DynamicPattern>

  <asp:PlaceHolder runat="server" id="itemPlaceholder" />

  <desDD:DynamicPattern id="DPFooterRow" runat="server" AutoGenerateButtons="Last"
    PatternTemplateName="ListOfDetailViews" PartsToGenerate="FooterRow" >

  <desDD:DynamicPattern ID="DPClosingTag" runat="server"
    PatternTemplateName="ListOfDetailViews" PartsToGenerate="ContainerClosingTag" />
</GroupTemplate>

<EmptyItemTemplate>
  <desDD:DynamicPattern id="DPReadOnly" runat="server"
    PatternTemplateName="ListOfDetailViews" PartsToGenerate="EmptyItem"
    Mode="ReadOnly" >
  </desDD:DynamicPattern>
</EmptyItemTemplate>

<ItemSeparatorTemplate>
  <desDD:DynamicPattern id="DPEdit" runat="server"
    PatternTemplateName="ListOfDetailViews" PartsToGenerate="DataRowSeparator" >
  </desDD:DynamicPattern>
</ItemSeparatorTemplate>

Get these from the previous example:
<ItemTemplate>, <EditItemTemplate>, <InsertItemTemplate>,<SelectedItemTemplate>,
<AlternatingItemTemplate>, and <EmptyDataTemplate>

</asp:ListView>
```

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

Example: DynamicListView

The DynamicListView establishes a default for **AutoGenerateButtons** as `First`.

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
   DataSourceID="DataSourceID" PatternTemplateName="ListOfDetailViews" >
</desDD:DynamicListView>
```

Example: DynamicListView using Groups

The DynamicListView establishes a default for **AutoGenerateButtons** as `First`.

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
   DataSourceID="DataSourceID" PatternTemplateName="ListOfDetailViews"
   EnableGrouping="True" ShowDataRowSeparator="True"
   DataRowSeparatorNamedPart="GroupAsMultipleColumns" GroupItemsCount="1" >
</desDD:DynamicListView>
```

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

NamedStyles on the ListOfDetailViews PatternTemplate

Add `PeterBlum.DES.DynamicData.NamedStyle` objects to the **NamedStyles** property on the DynamicPattern control to declare any of these style names. The style sheet classes are declared in the **[web app]/DynamicData/Content/PatternTemplateNamedStyles.css** file.

NamedStyle	Usage	Default style sheet class
"Container"	Style for the container tag around the entire data bound control.	"ListOfDetailViewsContainer"
"DataRow"	Style for the container tag for each data row.	"ListOfDetailViewsDataRow"
"SelectedRow"	Style for the container tag for the selected data row.	"ListOfDetailViewsSelectedRow"
"AlternativeRow"	Style for the container tag for alternative data rows.	"ListOfDetailViewsAlternativeRow"
"DataCell"	Style for the container tag around cells in data rows.	"ListOfDetailViewsDataCell"
"DataColumn"	Style for the container tag on the data column.	"ListOfDetailViewsDataColumn"
"DataControl"	Style for any DynamicControl.	"ListOfDetailViewsDataControl"
"LabelColumn"	Style for the container tag on the label column.	"ListOfDetailViewsLabelColumn"
"LabelControl"	Style for DynamicLabel control	"ListOfDetailViewsLabelControl"
"ButtonsCell"	Style for columns (`<td>` tag) that contain Buttons. If not defined, it uses "DataColumn".	"ListOfDetailViewsButtonsCell"
"DataRowSeparator"	Style for the container tag around a DataRow Separator.	"ListOfDetailViews-DataRowSeparator"

Example

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
   PatternTemplateName="ListOfDetailViews" >
   <NamedStyles>
      <desDD:NamedStyle Name="LabelColumn" CssClass="MyClass" />
      <desDD:NamedStyle Name="DataColumn" Style="width:200px;" />
   </NamedStyles>
</desDD:DynamicPattern>
```

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

NamedParts on the ListOfDetailViews PatternTemplate

There are additional Templates in this file under the `<NamedParts>` nodes of various Parts. When used, you get an alternative user interface that can show just about anything.

NamedParts can be used in the following ways:

- A default NamedPart name can be specified in properties on the DynamicPattern and DynamicListView controls. Each property has this format: ***PartType*NamedPart**. For example, to use the "Buttons" named part, use ButtonsNamedPart="Buttons".
- Assign the name to the **NamedPart** property on an ItemInPattern object (DataFieldInPattern, ButtonsInPattern, DataFieldsListInPattern, etc.)

Here are the predefined NamedParts in the ListOfDetailViews Pattern Template. Feel free to add your own or customize these:

Name/PartType	Description
"Buttons" in `<Buttons>`	Uses the **ButtonsForList.ascx** Pattern Template file, which provides a table row with the appropriate buttons for use in Read Only, Edit, and Insert modes. Standard Buttons are used.
"LinkButtons" in `<Buttons>`	Uses the **LinkButtonsForList.ascx** Pattern Template file, which provides a table row with the appropriate buttons for use in Read Only, Edit, and Insert modes. LinkButtons are used.
"SelectButton" in `<Buttons>`	Shows only single button for the Select command. Standard Buttons are used. Choose this when the DynamicControlPanel contains a DynamicButtonsField to offer the rest of the buttons.
"LinkSelectButton" in `<Buttons>`	Same as "SelectButton" but using LinkButtons.
"GroupAsMultipleColumns" in `<DataRowSeparator>`	Used with the Grouping feature (EnableGrouping=true on DynamicListView), this separates rows when using the default <GroupContainer>, <GroupHeader>, <GroupFooter>, <EmptyItem>, and <GroupSeparator> Parts

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

Using TwoColumnTableRow.ascx PatternTemplate

TwoColumnTableRow.ascx is a Pattern Template subclassed from `PeterBlum.DES.DynamicData.DataFieldsPatternTemplate`. It mimics the appearance of a row in the DetailsView web control.

```
<tr>
   <td>[DynamicLabel]</td>
   <td>[DynamicControl]</td>
</tr>
```

Usage Example

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
   PatternTemplateName="TwoColumnTableRow" DataField="columnname1" />
```

Because it is not a PartsPatternTemplate, you can ignore the **PartsToGenerate** property. Still use its **Mode** property to determine the data entry mode on Data Rows. You can usually ignore the **ItemsInPattern** property and assign its **DataField** property instead because it only needs one data field.

NamedStyles on the TwoColumnTableRow PatternTemplate

Add `PeterBlum.DES.DynamicData.NamedStyle` objects to the **NamedStyles** property on the DynamicPattern control to declare any of these style names. The style sheet classes are declared in the **[web app]/DynamicData/Content/PatternTemplateNamedStyles.css** file.

NamedStyle	Usage	Default Style Sheet Class
"LabelColumn"	Style for the first column (`<td>` tag), containing the label.	"TCTR_LabelColumn"
"LabelControl"	Style for the DynamicLabel control in the first column.	"TCTR_LabelControl"
"DataColumn"	Style for the second column (`<td>` tag), containing the data.	"TCTR_DataColumn"
"DataControl"	Style for the DynamicControl in the second column.	"TCTR_DataControl"
"Row"	Style for the TR tag.	"TCTR_Row"

Example

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
   PatternTemplateName="TwoColumnTableRow" DataField="columnname1" >
   <NamedStyles>
      <desDD:NamedStyle Name="LabelColumn" CssClass="MyClass" />
      <desDD:NamedStyle Name="DataColumn" Style="width:200px;" />
   </NamedStyles>
</desDD:DynamicPattern>
```

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

Using TwoStacked.ascx PatternTemplate

TwoStacked.ascx is a Pattern Template subclassed from `PeterBlum.DES.DynamicData.DataFieldsPatternTemplate`. It displays two data fields, separated by a `<br />` tag.

```
[DynamicLabel1] [DynamicControl1]<br />
[DynamicLabel2] [DynamicControl2]<br />
```

Usage Example

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
     PatternTemplateName="TwoStacked">
   <ItemsInPattern>
      <desDD:DataFieldInPattern DataField="columnname1" />
      <desDD:DataFieldInPattern DataField="columnname2" />
   </ItemsInPattern>
</desDD:DynamicPattern>
```

NamedStyles on the TwoStacked PatternTemplate

Add `PeterBlum.DES.DynamicData.NamedStyle` objects to the **NamedStyles** property on the DynamicPattern control to declare any of these style names. The style sheet classes are declared in the **[web app]/DynamicData/Content/PatternTemplateNamedStyles.css** file.

NamedStyle	Usage	Default Style Sheet Class
"LabelControl"	Style for the first DynamicLabel control.	"TwoStackedLabelControl"
"DataControl"	Style for the first DynamicControl.	"TwoStackedDataControl"
"LabelControl2"	Style for the second DynamicLabel control. If not declared, use "LabelControl".	"TwoStackedLabelControl2"
"DataControl2"	Style for the second DynamicControl. If not declared, use "DataControl".	"TwoStackedDataControl2"

Example

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
     PatternTemplateName="TwoStacked">
   <desDD:DataFieldInPattern DataField="columnname1" />
   <desDD:DataFieldInPattern DataField="columnname2" />
   <NamedStyles>
      <desDD:NamedStyle Name="LabelControl" CssClass="MyClass" />
      <desDD:NamedStyle Name="DataControl" Style="width:200px;" />
   </NamedStyles>
</desDD:DynamicPattern>
```

Properties on the TwoStacked PatternTemplate

To set these properties, see "Passing properties to be used by the PatternTemplate's own code".

- **ShowLabels** (Boolean) – Determines if the DynamicLabels are shown or hidden. When hidden, only the DynamicControls are shown.

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

Using DetailsViewButtons.ascx and DetailsViewLinkButtons.ascx PatternTemplates

DetailsViewButtons.ascx and **DetailsViewLinkButtons.ascx** are Pattern Template subclassed from `PeterBlum.DES.DynamicData.DynamicButtonsPatternTemplate`. They are designed to mimic buttons shown in a DetailsView control. **DetailsViewButtons.ascx** is shows Button controls while **DetailsViewLinkButtons.ascx** shows LinkButtons controls. Internally this Pattern Template uses a DynamicButtons control to create the buttons.

You can use these Pattern Templates either through a DynamicPattern control or when using the **DetailsView.ascx** Pattern Template, specifying the name “Buttons” or “LinkButtons” in its **ButtonsNamedPart** property.

It outputs this format in ReadOnly mode.

```
<tr>
  <td>[Delete button]</td>
  <td>[Edit button] [Insert button]</td>
</tr>
```

It outputs this format in Edit and Insert modes.

```
<tr>
  <td> </td>
  <td>[Save button] [Cancel button]</td>
</tr>
```

Each button has its **CommandName** property set to the standard commands of ListView and FormView: "Delete", "Edit", "New", "Update", "Insert" or "Cancel".

You can modify the HTML. In addition, you can change the properties of the buttons, especially taking advantage of DES's special properties on buttons. The Customizer control can also set DES’s special properties on buttons.

Usage Example: In a DynamicPattern

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
  PatternTemplateName="DetailsViewButton" />
```

Usage Example: Using a DetailsView PatternTemplate

DetailsView.ascx uses the NamedPart called “Buttons” or “LinkButtons” to load these Pattern Templates.

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
  PatternTemplateName="DetailsView" ButtonsPartName="Buttons" >
</desDD:DynamicPattern>
```

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

NamedStyles on these PatternTemplates

Add `PeterBlum.DES.DynamicData.NamedStyle` objects to the **NamedStyles** property on the DynamicPattern control to declare any of these style names. The style sheet classes are declared in the **[web app]/DynamicData/Content/PatternTemplateNamedStyles.css** file.

NamedStyle	Usage	Default Style Sheet Class
"ButtonLabelColumn"	Style for the first column (`<td>` tag).	"DetailsViewButtonLabelColumn"
"ButtonDataColumn"	Style for the second column (`<td>` tag).	"DetailsViewButtonDataColumn"
"ButtonRow"	Style for the `<tr>` tag	"DetailsViewButtonRow"
"ButtonControl"	Style applied to individual buttons	None by default

Example: In a DynamicPattern

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
   PatternTemplateName="DetailsViewButtons" >
   <NamedStyles>
      <desDD:NamedStyle Name="ButtonRow" CssClass="MyClass" />
      <desDD:NamedStyle Name="ButtonLabelControl" Style="width:200px;" />
   </NamedStyles>
</desDD:DynamicPattern>
```

Example: Using a DetailsView PatternTemplate

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
   PatternTemplateName="DetailsView" ButtonsPartName="Buttons" >
   <NamedStyles>
      <desDD:NamedStyle Name="ButtonRow" CssClass="MyClass" />
      <desDD:NamedStyle Name="ButtonLabelControl" Style="width:200px;" />
   </NamedStyles>
</desDD:DynamicPattern>
```

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

Using ButtonsForList.ascx and LinkButtonsForList.ascx PatternTemplates

ButtonsForList.ascx and **LinkButtonsForList.ascx** are Pattern Template subclassed from `PeterBlum.DES.DynamicData.DynamicButtonsPatternTemplate`. They are designed for buttons in a List style interface and work well with **GridView.ascx** and **ListOfDetailViews.ascx** Pattern Templates. **ButtonsForList.ascx** is shows Button controls while **LinkButtonsForList.ascx** shows LinkButtons controls.

You can use these Pattern Templates directly in DynamicPattern control. They can be used from DynamicListView when using the **GridView.ascx** and **ListOfDetailViews.ascx** Pattern Templates by specifying the name "Buttons" or "LinkButtons" in its **ButtonsNamedPart** property.

It outputs this format in ReadOnly mode.

```
<td>[Edit button] [Delete button] [Select button] 
[Details button]</td>
```

It outputs this format in Edit and Insert modes.

```
<td>[Save button] [Cancel button]</td>
```

Each button has its **CommandName** property set to the standard commands of ListView and FormView: "Delete", "Edit", "New", "Select", "Details", "Update", "Insert" or "Cancel".

You can modify the HTML. In addition, you can change the properties of the buttons, especially taking advantage of DES's special properties on buttons. The Customizer control can also set DES's special properties on buttons.

Usage Example: In a DynamicPattern

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
   PatternTemplateName="ButtonsForList" />
```

Usage Example: Using a GridView PatternTemplate

GridView.ascx uses the NamedPart called "Buttons" or "LinkButtons" to load these Pattern Templates.

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
   PatternTemplateName="GridView" ButtonsNamedPart="Buttons" >
</desDD:DynamicPattern>
```

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

NamedStyles on these PatternTemplates

Add `PeterBlum.DES.DynamicData.NamedStyle` objects to the **NamedStyles** property on the DynamicPattern control to declare any of these style names. The style sheet classes are declared in the **[web app]/DynamicData/Content/PatternTemplateNamedStyles.css** file.

NamedStyle	Usage	Default Style Sheet Class
"ButtonColumn"	Style for individual buttons	Nothing is assigned by default

Example: In a DynamicPattern

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
   PatternTemplateName="ButtonsForList" >
   <NamedStyles>
      <desDD:NamedStyle Name="ButtonColumn" CssClass="MyClass" />
   </NamedStyles>
</desDD:DynamicPattern>
```

Example: Using a GridView PatternTemplate

GridView.ascx uses the NamedPart called "Buttons" or "LinkButtons" to load these Pattern Templates.

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
   PatternTemplateName="GridView" ButtonsNamedPart="Buttons" >
   <NamedStyles>
      <desDD:NamedStyle Name="ButtonColumn" CssClass="MyClass" />
   </NamedStyles>
</desDD:DynamicPattern>
```

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

Using ButtonsForSingle.ascx and LinkButtonsForSingle.ascx PatternTemplates

ButtonsForSingle.ascx and **LinkButtonsForSingle.ascx** are Pattern Template subclassed from `PeterBlum.DES.DynamicData.DynamicButtonsPatternTemplate`. They are designed for buttons in a List style interface and work well with **DetailsView.ascx** Pattern Template. **ButtonsForSingle.ascx** is shows Button controls while **LinkButtonsForList.ascx** shows LinkButtons controls.

You can use these Pattern Templates directly in DynamicPattern control. They can be used from DynamicListView when using the **DetailsView.ascx** Pattern Template by specifying the name "Buttons" or "LinkButtons" in its **ButtonsNamedPart** property.

It outputs this format in ReadOnly mode.

```
<td>[Delete button] [Edit button] [New button]</td>
```

It outputs this format in Edit and Insert modes.

```
<td>[Save button] [Cancel button]</td>
```

Each button has its **CommandName** property set to the standard commands of ListView and FormView: "Delete", "Edit", "New", "Select", "Details", "Update", "Insert" or "Cancel".

You can modify the HTML. In addition, you can change the properties of the buttons, especially taking advantage of DES's special properties on buttons. The Customizer control can also set DES's special properties on buttons.

Usage Example: In a DynamicPattern

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
   PatternTemplateName="ButtonsForSingle" />
```

Usage Example: Using a DetailsView PatternTemplate

DetailsView.ascx uses the NamedPart called "ButtonsTogether" or "LinkButtonsTogether" to load these Pattern Templates.

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
   PatternTemplateName="DetailsView" ButtonsNamedPart="Buttons" >
</desDD:DynamicPattern>
```

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

NamedStyles on these PatternTemplates

Add `PeterBlum.DES.DynamicData.NamedStyle` objects to the **NamedStyles** property on the DynamicPattern control to declare any of these style names. The style sheet classes are declared in the **[web app]/DynamicData/Content/PatternTemplateNamedStyles.css** file.

NamedStyle	Usage	Default Style Sheet Class
"ButtonColumn"	Style for individual buttons	Nothing is assigned by default

Example: In a DynamicPattern

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
   PatternTemplateName="ButtonsForSingle" >
   <NamedStyles>
      <desDD:NamedStyle Name="ButtonColumn" CssClass="MyClass" />
   </NamedStyles>
</desDD:DynamicPattern>
```

Example: Using a DetailsView PatternTemplate

DetailsView.ascx uses the NamedPart called "ButtonsTogether" or "LinkButtonsTogether" to load these Pattern Templates.

```
<desDD:DynamicPattern ID="DynamicPattern1" runat="server"
   PatternTemplateName="DetailsView" ButtonsNamedPart="Buttons" >
   <NamedStyles>
      <desDD:NamedStyle Name="ButtonColumn" CssClass="MyClass" />
   </NamedStyles>
</desDD:DynamicPattern>
```

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

Customizing and creating your own PatternTemplates

A key feature Pattern Templates is that it allows modification to the HTML it generates. You are expected to tweak the existing Pattern Templates and create others.

Before you do, determine if an existing Pattern Template is correct except for a style (like style sheet class, width, or preventing word wrap) or formatting attribute (like cell alignment on a `<td>` tag). If that is the case, review the Named Styles of the Pattern Template to see if any are assigned to the tags you want to control. Then create an entry in the **NamedStyles** property of the DynamicPattern, DynamicListView, or DynamicFormView control that adjusts the style or formatting. See "Named Styles: Your pages control the formatting". This can eliminate modifying the Pattern Template.

There are several approaches to customizing a Pattern Template:

- Edit an existing file directly. This will effect all DynamicPattern, DynamicListView, and DynamicFormView controls that reference the Pattern Template. If the Pattern Template is a PartsPatternTemplate file, you can create new NamedParts that will not interfere with existing code.
- Duplicate the existing file, giving it a new name. Edit the duplicate. Assign the new file name to the **PatternTemplateName** property on the DynamicPattern, DynamicListView, or DynamicFormView control.
- Copy the existing file into a separate folder. Edit the copy. Assign the URL to the folder to the **PatternTemplatesFolderUrl** property on the DynamicPattern, DynamicListView, or DynamicFormView control. Assign the file name to the property as shown in the previous item.
- Create a new Pattern Template file. The next few sections will explore this option.

Click on any of these topics to jump to them:

- The format of PatternTemplate files
- Creating your PatternTemplate with the DataFieldsPatternTemplate class
- Creating your PatternTemplate with the PartsPatternTemplate class
- Creating your PatternTemplate with the DynamicButtonsPatternTemplate class
- Creating your PatternTemplate with the NoDataFieldsPatternTemplate class
- Supporting Named Styles within your PatternTemplates

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

The format of PatternTemplate files

A Pattern Template is a UserControl file. The file should have the extension ".ascx". The recommended folder for the file is **[web app]/DynamicData/PatternTemplates**.

The UserControl file should inherit from one of the following classes in the `PeterBlum.DES.DynamicData` namespace, within its `<%@ Control %>` tag.

ClassName	Usage
DataFieldsPatternTemplate	For simple layouts. It generally has one or more DynamicControls. Used by **TwoColumnTableRow.ascx** and **TwoStacked.ascx**.
PartsPatternTemplate	For Pattern Templates that use the DynamicPartsPatterns control. It defines all of the "Parts" of a complete data bound control, like DataRow, HeaderRow, and NoDataToShow. Its "NamedParts" feature allows alternative layouts for each Parts. See "The DynamicPartsPatterns Control". Used by **DetailsView.ascx**, **GridView.ascx**, and **ListOfDetailViews.ascx**.
NoDataFieldsPatternTemplate	Pattern Templates do not contain a DynamicControl or DynamicLabel. This class lets you establish HTML and web controls that still use the Named Styles and Attribute features on the DynamicPattern control.
DynamicButtonsPatternTemplate	A Pattern Template built around the DynamicButtons control to offer a standard user interface for buttons. It describes the formatting of buttons for each data entry mode. Used by **DetailsViewButton.ascx**, **DetailsViewLinkButton.ascx**, **ButtonsForList.ascx** or **LinkButtonsForList.ascx**.

These classes are derived from `PeterBlum.DES.DynamicData.BasePatternTemplate`, an abstract class.

Examples

```
<%@ Control Language="C#"
   Inherits="PeterBlum.DES.DynamicData.DataFieldsPatternTemplate" %>

<%@ Control Language="VB"
   Inherits="PeterBlum.DES.DynamicData.PartsPatternTemplate" %>
```

The remainder of the file depends on the HTML you wish to generate and any requirements imposed by the ancestor Pattern Template class. Click on the appropriate class name in the table above for specifics.

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

Creating your PatternTemplate with the DataFieldsPatternTemplate class

The DataFieldsPatternTemplate handles the "classic" case for a template, and is much like the concept of the "Entity Template" that is in ASP.NET Dynamic Data v4.0. It describes the layout of a single pattern, including HTML and one or more DynamicControls. It is typically used to handle a part of an `<ItemTemplate>` in the ListView or FormView, such as all of the formatting for a single data field.

DES Dynamic Data includes two examples of the DataFieldsPatternTemplate within your **[web app]/DynamicData/PatternTemplates** folder: **TwoColumnTableRow.ascx** and **TwoStacked.ascx**. When creating your own, explore these files as sample code.

Steps for creating a DataFieldsPatternTemplate

Click on any of these topics to jump to them:

- Part 1: Create the UserControl file
- Part 2: Create the controls
- Part 3: Supporting Named Styles
- Part 4: Support customization

Part 1: Create the UserControl file

1. Create a UserControl file. Typically its added to the **[web app]/DynamicData/PatternTemplates** folder, but it can go anywhere in your web app, so long as you set the **PatternTemplateFolderUrl** property on the DynamicPattern control.

   Suggestion: Don't use the code behind file especially if you plan on duplicating this file. Its easier to manage duplicating with a single file.

2. Modify its `<%@ Control %>` tag to inherit from `PeterBlum.DES.DynamicData.DataFieldsPatternTemplate`.

```
<%@ Control Language="Language option"
 Inherits="PeterBlum.DES.DynamicData.DataFieldsPatternTemplate" %>
```

3. *Optional.* You can assist design mode by adding the **Usage** property to the `<%@ Control %>` tag. It lets the Properties Editor know your intended usage for the Pattern Template, so it shows up only in the appropriate properties.

```
<%@ Control Language="Language option" Usage="Data"
 Inherits="PeterBlum.DES.DynamicData.DataFieldsPatternTemplate" %>
```

   The **Usage** property takes one of these values:

   - `Any` – It will always appear. If the **Usage** property is not specified, this is the default.
   - `Buttons` – It represents a button, and is commonly found in ButtonPatternTemplates.
   - `Data` – It will contain data oriented controls, such as the DynamicControl.
   - `Header` – It will appear in a header row, such as a column title.
   - `Footer` – It will appear in a footer row.

STEPS CONTINUE ON THE NEXT PAGE

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

Part 2: Create the controls

4. Define the HTML and controls.

   - Use DynamicControls wherever a Field Template should be setup. Do not assign its **DataField** property. (If you do, that DynamicControl will only use the data field name given here.). Give the control a useful ID as the DynamicControl may set up the **DataField** by the ID of the control.
   - Use DynamicLabel controls wherever a label is needed. Again do not assign the **DataField** property and again, give it a useful ID. If this DynamicLabel is associated with a DynamicControl also contained in this template, set the **AssociatedControlID** property to that DynamicControl's ID. *ALERT: Use only DES's DynamicLabel, not the same-named control that comes any release of ASP.NET Dynamic Data.*
   - Establish the HTML formatting and other controls.

5. *Optional.* At this point, the DataFieldsPatternTemplate knows how to locate your DynamicControls and DynamicLabels. It does so through a search of the control tree. It creates a list of those found. The DynamicPattern control will later refer to these controls either by its position in the list (starting at 0) or by its ID.

   You may want to take more control of this to speed up the search or change the order. Override the `RegisterControls()` method which is declared in the BasePatternTemplate class. Your code will establish the same list of controls, in the order of calls to the `RegisterDynamicControl()` and `RegisterDynamicLabel()`methods.

   Here is an example that registers two DynamicControls and two DynamicLabels.

   [C#]

```
public override void RegisterControls()
{
   RegisterDynamicControl(DynamicControl1);
   RegisterDynamicLabel(DynamicLabel1);
   RegisterDynamicControl(DynamicControl2);
   RegisterDynamicLabel(DynamicLabel2);
}
```

   [VB]

```
Public Overrides Sub RegisterControls()
   RegisterDynamicControl(DynamicControl1)
   RegisterDynamicLabel(DynamicLabel1)
   RegisterDynamicControl(DynamicControl2)
   RegisterDynamicLabel(DynamicLabel2)
End Sub
```

STEPS CONTINUE ON THE NEXT PAGE

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

Part 3: Supporting Named Styles

For an overview of Named Styles, see "Named Styles: Your pages control the formatting".

6. Identify any HTML elements or controls that may allow the web form developer to override its styles and formatting within the DynamicPattern control. What to consider:

   - Tags and controls that may get set to a style sheet class.
   - Elements that can determine the width or height. Page developers may want to adjust their height.
   - Table cells, as they can have their alignment overridden.

   If you have no need for Named Styles, you are finished at this step.

7. Define any Named Styles based on the list developed in the previous step by defining a name and description for each. It will help to include code-level documentation with this list, such as this C# code from **TwoStacked.ascx**:

```
/* ---- USAGE -------------------------------------------------------------
    … comments about usage here …
---- NAMED STYLES -------------------------------------------------------
 Use the DynamicPattern.NamedStyles property to declare any of these style names.

   "LabelControl" - Style for the first DynamicLabel control.
   "DataControl" - Style for the first DynamicControl.
   "LabelControl2" - Style for the second DynamicLabel. If not declared, it uses "LabelControl".
   "DataControl2" - Style for the second DynamicControl. If not declared, it uses "DataControl".
```

8. Convert any HTML tag that will use a Named Style to a DynamicStylesTag control. Assign these properties:

   - **Tag** – The tag name.
   - **CssClass** – The `class=` attribute value
   - **Style** – The `style=` attribute value
   - For any other attributes, leave them in place as properties on the DynamicStylesTag.

   In this example, the `<td>` tag is converted.

   Original

```
<td class="LabelColumn" style="text-align:left;white-space:nowrap;"
  valign="top" >
  <desDD:DynamicLabel id="DynamicLabel1" runat="server"
    AssociatedControlID="DynamicControl1" CssClass="LabelControl" />
</td>
```

   Using DynamicStylesTags

```
<desDD:DynamicStylesTag runat="server" NamedStyle="LabelColumn"
  Tag="Td" CssClass="LabelColumn" Style="text-align:left;white-space:nowrap;"
  valign="top" >
  <desDD:DynamicLabel id="DynamicLabel1" runat="server"
    AssociatedControlID="DynamicControl1" CssClass="LabelControl" />
</desDD:DynamicStylesTag>
```

   For more detail on the DynamicStylesTag, see "The DynamicStylesTag Control".

   Note: You can use the GetTagAttributes() method in the next step or the ApplyNamedStyles() method described later to do the same work as the DynamicStylesTag.

STEPS CONTINUE ON THE NEXT PAGE

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

9. If you have an opening tag without a closing tag that needs Named Style support, declare the actual HTML tag and embed the DynamicStylesTag control within the HTML tag's header. Do not assign the **Tag** property.

```
<tr <desDD:DynamicStylesTag runat="server" NamedStyle="DataRow"
  CssClass="DataRow" > >
```

Alternatively embed a call to the `GetTagAttributes()` method (part of the PatternTemplate base class) within ASP tags like this:

```
<tr<%=GetTagAttributes("DataRow", "DataRow") %>>
```

The first parameter is the default style sheet class. It can be an empty string. The second parameter is the name of the Named Style. There are several versions of `GetTagAttributes()`. See "The PatternTemplate.GetTagAttributes method".

10. For any remaining control that needs a Named Style, call the `ApplyNamedStyle()` method in the `Page_Load()` method of your UserControl. See "The PatternTemplate.ApplyNamedStyle method".

Here is an example that establishes the Named Styles for **TwoStacked.ascx**, which has two DynamicControls and two DynamicLabels:

[C#]

```
protected void Page_Load(object pSender, EventArgs pE)
{
   ApplyNamedStyle("LabelControl", DynamicLabel1);
   ApplyNamedStyle("DataControl", DynamicControl1);
   ApplyNamedStyle("LabelControl2", "LabelControl", DynamicLabel2);
   ApplyNamedStyle("DataControl2", "DataControl", DynamicControl2);
}
```

[VB]

```
Protected Sub Page_Load(ByVal pSender As Object, ByVal pE As EventArgs)
   ApplyNamedStyle("LabelControl", DynamicLabel1)
   ApplyNamedStyle("DataControl", DynamicControl1)
   ApplyNamedStyle("LabelControl2", "LabelControl", DynamicLabel2)
   ApplyNamedStyle("DataControl2", "DataControl", DynamicControl2)
End Sub
```

11. *Optional.* Design mode can display the list of Named Styles in the Properties Editor if you add an XML block to your UserControl. It belongs in the C# or VB code area and must be commented out. (The PatternTemplate parser will know to ignore VB's REM statements.)

The XML has this pattern:

```
<NamedStylesDoc>
   <NamedStyle Name="Name1">description</NamedStyle>
   <NamedStyle Name="Name2">description</NamedStyle>
      [and more]
</NamedStylesDoc>
```

Use any of the Pattern Templates shipped with DES for examples.

STEPS CONTINUE ON THE NEXT PAGE

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

Part 4: Support customization

12. *Optional*. Create public properties that allow the user to customize the behavior of your Pattern Template. Your properties must retrieve values from the DynamicPattern control by using `GetAttributeAsNative()`, `GetAttributeAsInteger()`, `GetAttributeAsBoolean()`, `GetAttributeAsUnit()`, or `GetAttributeAsString()` methods. See "Passing properties to be used by the PatternTemplate's own code".

    When coding, generally use the `OnPreRender()` method to change the appearance and visibility of your controls.

Example

Suppose you have this block of HTML and controls, which shows up to 2 DynamicControls. One will always show. The other is contained in a PlaceHolder which is made visible only when the **ShowSecondField** property is set to `true`.

[C#]

```
<%@ Control Language="C#" Usage="Data"
    Inherits="PeterBlum.DES.DynamicData.DataFieldsPatternTemplate" %>

<script runat="server">
// When true, show the second data field
public bool ShowSecondField
{
  get { return Host.GetAttributeAsBool("ShowSecondField",
     cDefaultShowSecondField); }
}
private const bool cDefaultShowSecondField = false;

protected override OnPreRender(EventArgs pArgs)
{
   base.OnPreRender(pArgs);
   PlaceHolder1.Visible = ShowSecondField;
}
</script>

<desDD:DynamicLabel id="FirstLabel" runat="server"
   AssociatedControlID="FirstField" /> 
<desDD:DynamicControl id="FirstField" runat="server" />

<asp:PlaceHolder id="PlaceHolder1" runat="server">
   <br />
   <desDD:DynamicLabel id="SecondLabel" runat="server"
      AssociatedControlID="SecondField" /> 
   <desDD:DynamicControl id="SecondField" runat="server" />
</asp:PlaceHolder>
```

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

[VB]

```
<%@ Control Language="VB" Usage="Data"
    Inherits="PeterBlum.DES.DynamicData.DataFieldsPatternTemplate" %>

<script runat="server">
' When true, show the second data field
Public ReadOnly Property ShowSecondField () As Boolean
  Get
    Return Host.GetAttributeAsString("ShowSecondField",
      cDefaultShowSecondField)
  End Get
End Property
Private Const cDefaultShowSecondField As Boolean = False

Protected Overridden Sub OnPreRender(ByVal pArgs As EventArgs)
   MyBase.OnPreRender(pArgs)
   PlaceHolder1.Visible = ShowSecondField
End Sub
</script>

<desDD:DynamicLabel id="FirstLabel" runat="server"
   AssociatedControlID="FirstField" /> 
<desDD:DynamicControl id="FirstField" runat="server" />

<asp:PlaceHolder id="PlaceHolder1" runat="server">
   <br />
   <desDD:DynamicLabel id="SecondLabel" runat="server"
      AssociatedControlID="SecondField" /> 
   <desDD:DynamicControl id="SecondField" runat="server" />
</asp:PlaceHolder>
```

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

Creating your PatternTemplate with the PartsPatternTemplate class

The PartsPatternTemplate handles generates the HTML and controls for a list of data fields. It's the underlying technology of the **GridView.ascx**, **ListOfDetailViews.ascx**, and **DetailsView.ascx** Pattern Templates and is the only type of Pattern Template supported by the DynamicListView and DynamicFormView controls. When creating your own, explore these files as sample code.

The PartsPatternTemplate file requires a DynamicPartsPatterns control, which breaks up the DataBound control into its "Parts". The Parts you setup depend on the type of DataBound Control: list or single record.

For Lists, see "How to use the PartsPatternTemplate with a DynamicListView" and "Parts Overview for PartsPatternTemplate when used with Lists".

For Single Record, see "How to use the PartsPatternTemplate with a DynamicFormView" and "Parts Overview for PartsPatternTemplate when used with Single Records".

Steps for creating a PartsPatternTemplate

Click on any of these topics to jump to them:

- Part 1: Create the UserControl file
- Part 2: Create the controls
- Part 3: Supporting Named Styles
- Part 4: Support customization

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

Part 1: Create the UserControl file

1. Create a UserControl file. Typically its added to the **[web app]/DynamicData/PatternTemplates** folder, but it can go anywhere in your web app, so long as you set the **PatternTemplateFolderUrl** property on the DynamicPattern control.

   Suggestion: Don't use the code behind file especially if you plan on duplicating this file. Its easier to manage duplicating with a single file.

2. Modify its `<%@ Control %>` tag to inherit from PeterBlum.DES.DynamicData.PartsPatternTemplate.

```
<%@ Control Language="Language option"
 Inherits="PeterBlum.DES.DynamicData.PartsPatternTemplate" %>
```

3. *Optional.* You can assist design mode by adding the **Usage** property to the `<%@ Control %>` tag. It lets the Properties Editor know your intended usage for the Pattern Template, so it shows up only in the appropriate properties.

```
<%@ Control Language="Language option" Usage="PartsForList"
 Inherits="PeterBlum.DES.DynamicData.PartsPatternTemplate" %>
```

   The **Usage** property takes one of these values:

   - `Parts` – Used by PartsPatternTemplates and supported by DynamicListView/DynamicFormView.
   - `PartsForList` – Used by PartsPatternTemplates and supported by DynamicListView. Describes a list style interface.
   - `PartsForSingle` – Used by PartsPatternTemplates and supported by DynamicFormView. Describes a single record style interface.

STEPS CONTINUE ON THE NEXT PAGE

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

Part 2: Create the controls

4. Add the DynamicPartsPatterns control. All remaining HTML and controls will go inside of its Part nodes.

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
</desDD:DynamicPartsPatterns>
```

5. Override the `GetDynamicPartsPatterns()` method. Its only task is to return a reference to the DynamicPartsPatterns control.

[C#]

```
protected override DynamicPartsPatterns GetDynamicPartsPatterns()
{
   return DynamicPartsPattern1;
}
```

[VB]

```
Protected Overridden Function GetDynamicPartsPatterns() _
      As DynamicPartsPatterns
   Return DynamicPartsPattern1
End Function
```

6. Define the HTML and controls using the Part properties shown in these links.

- Parts Overview
- Properties of the DynamicPartsPatterns Control

Here are some guidelines:

- Use DynamicControls wherever a Field Template should be setup. Do not assign its **DataField** property. (If you do, that DynamicControl will only use the data field name given here.). Give the control a useful ID as the DynamicControl may set up the **DataField** by the ID of the control.
- Use DynamicLabel controls wherever a label is needed. Again do not assign the **DataField** property and again, give it a useful ID. If this DynamicLabel is associated with a DynamicControl also contained in this template, set the **AssociatedControlID** property to that DynamicControl's ID. *ALERT: Use only DES's DynamicLabel, not the same-named control that comes any release of ASP.NET Dynamic Data.*
- Establish the HTML formatting and other controls.

Example Parts

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
   <DataCell NamedStyle="DataCell" CssClass="DataCellStyle"
         Style="vertical-align:top;" >

      <Template>
         <desDD:DynamicControl ID="DynamicControl1" runat="server" />
      </Template>

   </DataCell>

   <NoDataToShow Tag="Div" NamedStyle="NoDataToShow"
         CssClass="NoDataToShowStyle" Message="No records were found." >
   </NoDataToShow>

</desDD:DynamicPartsPatterns>
```

STEPS CONTINUE ON THE NEXT PAGE

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

7. Use the NamedParts feature of the DynamicPartsPatterns control to alternative cases for the Parts. Every Part provides a **NamedParts** property where you can define a list of alternatives.

   - NamedParts Overview
   - Properties of the DynamicPartsPatterns Control

   When defining them:

   - Assign the **Name** property with a unique value amongst this Part's list. *It can be identical to names in other Parts.*
   - Always use the correct class as shown in this table:

Part tag	NamedPart class
`<Container>`	ContainerNamedPatternPart
`<DataRow>`	DataRowNamedPatternPart
`<DataCell>`	DataCellNamedPatternPart
`<Buttons>`	ButtonsNamedPatternPart
`<HeaderRow>`	HeaderRowNamedPatternPart
`<HeaderCell>`	HeaderCellNamedPatternPart
`<HeaderButtons>`	HeaderButtonsNamedPatternPart
`<FooterRow>`	FooterRowNamedPatternPart
`<FooterCell>`	FooterCellNamedPatternPart
`<FooterButtons>`	FooterButtonsNamedPatternPart
`<DataRowSeparator>`	DataRowSeparatorNamedPatternPart
`<DataRowSeparatorCell>`	DataRowSeparatorCellNamedPatternPart
`<NoDataToShow>`	NoDataToShowNamedPatternPart
`<GroupContainer>`	GroupContainerNamedPatternPart
`<GroupHeader>`	GroupHeaderNamedPatternPart
`<GroupFooter>`	GroupFooterNamedPatternPart
`<GroupSeparator>`	GroupSeparatorNamedPatternPart
`<EmptyItem>`	EmptyItemNamedPatternPart
`<EmptyItemCell>`	EmptyItemCellNamedPatternPart

8. If you want to create the requested Part programmatically, override the `OnPartCreating()` method.

   See "Overriding the PartsPatternTemplate.OnPartCreating() method".

9. If you want to modify the contents of a Part after its created, override the `OnPartCreated()` method.

   See "Overriding the PartsPatternTemplate.OnPartCreated() method".

STEPS CONTINUE ON THE NEXT PAGE

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

Part 3: Supporting Named Styles

For an overview of Named Styles, see "Named Styles: Your pages control the formatting".

10. Identify any HTML elements or controls that may allow the web form developer to override its styles and formatting within the DynamicPattern. What to consider:

    - Tags and controls that may get set to a style sheet class.
    - Elements that can determine the width or height. Page developers may want to adjust their height.
    - Table cells, as they can have their alignment overridden.

    If you have no need for Named Styles, you are finished at this step.

11. Define any Named Styles based on the list developed in the previous step by defining a name and description for each. It will help to include code-level documentation with this list, such as this C# code from **TwoStacked.ascx**:

```
/* ---- USAGE ------------------------------------------------------------
    … comments about usage here …
---- NAMED STYLES -------------------------------------------------------
 Use the DynamicPattern.NamedStyles property to declare any of these style names.
   "LabelControl" - Style for the first DynamicLabel control.
   "DataControl" - Style for the first DynamicControl.
   "LabelControl2" - Style for the second DynamicLabel. If not declared, it uses "LabelControl".
   "DataControl2" - Style for the second DynamicControl. If not declared, it uses "DataControl".
```

12. Convert any HTML tag that will use a Named Style to a DynamicStylesTag control. Assign these properties:

    - **Tag** – The tag name.
    - **CssClass** – The `class=` attribute value
    - **Style** – The `style=` attribute value
    - For any other attributes, leave them in place as properties on the DynamicStylesTag.

    In this example, the `<td>` tag is converted.

Original

```
<td class="LabelColumn" style="text-align:left;white-space:nowrap;" >
   <desDD:DynamicLabel id="DynamicLabel1" runat="server"
      AssociatedControlID="DynamicControl1" CssClass="LabelControl" />
</td>
```

Using DynamicStylesTags

```
<desDD:DynamicStylesTag runat="server" NamedStyle="LabelColumn"
   Tag="Td" CssClass="LabelColumn" Style="text-align:left;white-space:nowrap;" >
   <desDD:DynamicLabel id="DynamicLabel1" runat="server"
      AssociatedControlID="DynamicControl1" CssClass="LabelControl" />
</desDD:DynamicStylesTag>
```

For more detail on the DynamicStylesTag, see "The DynamicStylesTag Control".

Note: You can use the GetTagAttributes() method in the next step or the ApplyNamedStyles() method described later to do the same work as the DynamicStylesTag.

STEPS CONTINUE ON THE NEXT PAGE

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

13. If you have an opening tag without a closing tag that needs Named Style support, declare the actual HTML tag and embed the DynamicStylesTag control within the HTML tag's header. Do not assign the **Tag** property.

```
<tr <desDD:DynamicStylesTag runat="server" NamedStyle="DataRow"
  CssClass="DataRow" > >
```

Alternatively embed a call to the `GetTagAttributes()` method (part of the PatternTemplate base class) within ASP tags like this:

```
<tr<%=GetTagAttributes("DataRow", "DataRow") %>>
```

The first parameter is the default style sheet class. It can be an empty string. The second parameter is the name of the Named Style. There are several versions of `GetTagAttributes()`. See "The PatternTemplate.GetTagAttributes method".

14. For any remaining control that needs a Named Style, use these methods from within the `OnPartCreated()` method:

- `ApplyNamedStyle()` method on PatternTemplate to apply a NamedStyle to a specific control.
- `UpdateStyleOnDynamicControls()` method on *pArgs* to apply a NamedStyle and style sheet class name to all DynamicControls that were generated from the Part or registered in your `OnPartCreating()` method.
- `UpdateStyleOnDynamicLabels()` method on *pArgs* to apply a NamedStyle and style sheet class name to all DynamicLabels controls that were generated from the Part or registered in your `OnPartCreating()` method.

15. *Optional.* Design mode can display the list of Named Styles in the Properties Editor if you add an XML block to your UserControl. It belongs in the C# or VB code area and must be commented out. (The PatternTemplate parser will know to ignore VB's REM statements.)

The XML has this pattern:

```
<NamedStylesDoc>
   <NamedStyle Name="Name1">description</NamedStyle>
   <NamedStyle Name="Name2">description</NamedStyle>
      [and more]
</NamedStylesDoc>
```

Use any of the Pattern Templates shipped with DES for examples.

STEPS CONTINUE ON THE NEXT PAGE

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

Part 4: Support customization

16. *Optional*. Create public properties that allow the user to customize the behavior of your Pattern Template. Utilize existing properties from the ancestor classes too. See "Passing properties to be used by the PatternTemplate's own code".

    When coding, generally override the `OnPartCreated()` method to change the appearance and visibility of your controls.

Example

Suppose you have this block of HTML and controls, which shows up to 2 DynamicControls. One will always show. The other is contained in a PlaceHolder which is made visible only when the **ShowSecondField** property is set to `true`.

[C#]

```
<%@ Control Language="C#" Usage="Data"
    Inherits="PeterBlum.DES.DynamicData.PartsPatternTemplate" %>

<script runat="server">
// When true, show the second data field
public bool ShowSecondField
{
  get { return Host.GetAttributeAsBool("ShowSecondField",
    cDefaultShowSecondField); }
}
private const bool cDefaultShowSecondField = false;

protected override void OnPartCreated(Control pContainer,
   PartCreatedEventArgs pArgs)
{
   base.OnPartCreated(pContainer, pArgs);
   if (pArgs.Mode == DataBoundControlMode.ReadOnly && !ShowSecondField)
      pContainer.FindControl("PlaceHolder1").Visible = false;
}
</script>

<ItemTemplate>
   <desDD:DynamicLabel id="FirstLabel" runat="server"
      AssociatedControlID="FirstField" /> 
   <desDD:DynamicControl id="FirstField" runat="server" />

   <asp:PlaceHolder id="PlaceHolder1" runat="server">
      <br />
      <desDD:DynamicLabel id="SecondLabel" runat="server"
         AssociatedControlID="SecondField" /> 
      <desDD:DynamicControl id="SecondField" runat="server" />
   </asp:PlaceHolder>
</ItemTemplate>
```

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

[VB]

```
<%@ Control Language="VB" Usage="Data"
    Inherits="PeterBlum.DES.DynamicData.ReeaterPatternTemplate" %>

<script runat="server">
' When true, show the second data field
Public ReadOnly Property ShowSecondField () As Boolean
  Get
    Return Host.GetAttributeAsString("ShowSecondField",
      cDefaultShowSecondField)
  End Get
End Property
Private Const cDefaultShowSecondField As Boolean = False

Protected Overrides Sub OnPartCreated(ByVal pContainer As Control,
   ByVal pArgs As PartCreatedEventArgs)

   MyBase.OnPartCreated(pContainer, pArgs)
   If (pArgs.Mode = DataBoundControlMode.ReadOnly) And Not ShowSecondField Then
      pContainer.FindControl("PlaceHolder1").Visible = False
   End If
End Sub
</script>

<ItemTemplate>
   <desDD:DynamicLabel id="FirstLabel" runat="server"
      AssociatedControlID="FirstField" /> 
   <desDD:DynamicControl id="FirstField" runat="server" />

   <asp:PlaceHolder id="PlaceHolder1" runat="server">
      <br />
      <desDD:DynamicLabel id="SecondLabel" runat="server"
         AssociatedControlID="SecondField" /> 
      <desDD:DynamicControl id="SecondField" runat="server" />
   </asp:PlaceHolder>
</ItemTemplate>
```

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

Overriding the PartsPatternTemplate.OnPartCreating() method

If you want to create the controls for the requested Parts programmatically, override the `OnPartCreating()` method on the PartsPatternTemplate class. It is called before generating content from the Part, allowing you to generate alternative content for the Part. Use it to add controls to the *Container* control passed in. The *Args* parameter provides a variety of details on the action, including the mode (readonly, edit, insert), and NamedPart name requested.

Your `OnPartCreating()` method can apply the content of an existing Part to the *Container* control by calling `pArgs.CreateFromPart()`. Then modify the controls that were created.

Here is the function definition.

[C#]

```
protected virtual bool OnPartCreating(Control pContainer,
     PeterBlum.DES.DynamicData.PartCreatingEventArgs pArgs)
```

[VB]

```
Protected Overridable Function OnPartCreating( _
  ByVal pContainer As Control, _
  ByVal pArgs As PeterBlum.DES.DynamicData.PartCreatingEventArgs) _
    As Boolean
```

Parameters

pContainer

The control to which you add your own controls.

When you add a DynamicControl, call `pArgs.RegisterDynamicControl(your control)`.

When you add a DynamicLabel, call `pArgs.RegisterDynamicLabel(your control)`.

The order they are added is the order used by the **PositionInPattern** property of the DataFieldInPattern object that is used to assign the data field to the **DataField** property of these controls.

Note: If you don't use the PositionInPattern property, you can avoid calling these Register methods and let DES search through the pContainer control, but that is still less efficient that calling these methods.

pArgs

The arguments. See "The PeterBlum.DES.DynamicData.PartCreatingEventArgs class".

Returns

When `true`, the Part should not generate its content. You have prepared the *pContainer* property.

When `false`, the Part still needs to generate its content.

SEE THE NEXT PAGE FOR AN EXAMPLE

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

Example

This `OnPartCreating()` creates a NamedPart with the name "TryMe" that adds one DynamicControl and DynamicLabel control separated by "` `".

[C#]

```
using DES=PeterBlum.DES.DynamicData;
...
public override bool OnPartCreating(Control pContainer,
   DES.PartCreatingEventArgs pArgs)
{
   if ((String.Compare(pArgs.NamedPartsName, "TryMe", true) == 0)
   {
      DES.DynamicControl vDC = new DES.DynamicControl();
      vDC.ID = "DynamicControl1";
      pContainer.Controls.Add(vDC);
      pArgs.RegisterDynamicControl(vDC);

      pContainer.Controls.Add(new LiteralControl(" "));

      DES.DynamicLabel vDL = new DES.DynamicLabel();
      vDL.ID = "DynamicLabel1";
      pContainer.Labels.Add(vDL);
      pArgs.RegisterDynamicLabel(vDL);

      return true;
   }
   return false;
}
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
...
Public Overrides Function OnPartCreating(ByVal pContainer As Control, _
   ByVal pArgs As DES.PartCreatingEventArgs) As Boolean

   If String.Compare(pArgs.NamedPartsName, "TryMe", True) = 0 Then
      DES.DynamicControl vDC = New DES.DynamicControl()
      vDC.ID = "DynamicControl1"
      pContainer.Controls.Add(vDC)
      pArgs.RegisterDynamicControl(vDC)

      pContainer.Controls.Add(New LiteralControl(" "))

      DES.DynamicLabel vDL = New DES.DynamicLabel()
      vDL.ID = "DynamicLabel1"
      pContainer.Labels.Add(vDL)
      pArgs.RegisterDynamicLabel(vDL)

      Return True
   End If
   Return False
End Sub
```

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

The PeterBlum.DES.DynamicData.PartCreatingEventArgs class

This is a parameter for the `OnPartCreating()` method of the PartsPatternTemplate class. Use `OnPartCreating()` to create controls programmatically instead (or in addition to) the Parts declared in the DynamicPartsPatterns control.

One powerful feature of this system is that the page development can pass parameters to your `OnPartCreating()` method. Each ItemInPattern class supports "Attributes", a list of name/value pairs. The name is a string and is used to name the parameter. The value can be a string or any other type of object.

Lets suppose that you declare a Named Part called "Sample" that is implemented entirely through the `OnPartCreating()` method. You document that it accepts the parameter "ShowAll" with values of "Yes" and "No". The user can do this:

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server">
<ItemsInPattern>
  <desDD:NoDataFieldsInPattern NamedPart="Sample" ShowAll="Yes" />
</ItemsInPattern>
</desDD:DynamicPattern>
```

Your `OnPartCreating()` method will call `GetAttributeAsNative()` or `GetAttributeAsString()` to retrieve the value.

Click on any of these topics to jump to them:

- Properties of PartCreatingEventArgs
- GetAttributeAsNative method
- GetAttributeAsString method
- RegisterDynamicControl method
- RegisterDynamicLabel method
- CreateFromPart method

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

Properties of PartCreatingEventArgs

The `PeterBlum.DES.DynamicData.PartCreatingEventArgs` class has these read-only properties:

- **PartType** (enum PeterBlum.DES.DynamicData.PatternPartType) – The type of Part requested.

  The enumerated type `PeterBlum.DES.DynamicData.PatternPartType` has these values:

  - `Container` – Container around the entire control.
  - `DataRow` – Uses Mode to determine Template, EditTemplate, or Insertemplate
  - `DataCell` – Uses Mode to determine Template, EditTemplate, or Insertemplate
  - `Buttons` – Buttons within a Data Row. Used when AutoGenerateButtons is setup
  - `HeaderRow` – Header row
  - `HeaderCell` – A single field within a HeaderRow
  - `HeaderButtons` – Header for Buttons within a Header Row. Used when AutoGenerateButtons is setup
  - `FooterRow` – Footer row
  - `FooterCell` – A single field within a FooterRow
  - `FooterButtons` – Footer for Buttons within a Footer Row. Used when AutoGenerateButtons is setup
  - `DataRowSeparator` – For the ItemSeparatorTemplate of the ListView, this is a row that separates each DataRow. It can contain the entire HTML or let DataRowSeparatorCells define the appearance for each cell associated with a DataFieldInPattern object.
  - `DataRowSeparatorCell` – For the ItemSeparatorTemplate of the ListView, this a cell within a row that separates each DataRow. It only appears when DataRowSeparator indicates its use. One is generated for each DataFieldInPattern object.
  - `NoDataToShow` – When there are no rows of data to show. Generally this is a message to that face.
  - `GroupContainer` – For ListView's Grouping feature (automatic GroupTemplate generation). Defines the HTML tag that encloses a group. It should not be used for any HTML content, such as a header or footer. They are handled by GroupHeader and GroupFooter.
  - `GroupHeader` – For ListView's Grouping feature (automatic GroupTemplate generation). HTML that separates the Group's opening tag from GroupContainer from the first Item of actual data. It is often used for titles. It does not support DynamicControls.
  - `GroupFooter` – For ListView's Grouping feature (automatic GroupTemplate generation). HTML that separates the Group's closing tag from GroupContainer from the last Item of actual data. It does not support DynamicControls.
  - `GroupSeparator` – For ListView's Grouping feature (automatic GroupTemplate generation). HTML that separates groups from each other. Associated with the ListView GroupSeparatorTemplate feature.
  - `EmptyItem` – For ListView's Grouping feature. It sets up the value of the EmptyItemTemplate property. EmptyItemTemplate defines the appearance where an item would normally go in a group, but there is no data for that item.It can contain the entire HTML or let EmptyItemCells define the appearance for each cell associated with a DataFieldInPattern object.
  - `EmptyItemCell` – For the EmptyItemTemplate of the ListView, this a cell within a row that describes an record without any data. It only appears when EmptyItem indicates its use. One is generated for each DataFieldInPattern object.
  - `Unknown` – The part is not associated with the DynamicPartsPatterns, such as from a TemplateNotInPattern object.

- **Mode** (enum System.Web.UI.WebControls.DataBoundControlMode) – The mode: ReadOnly, Edit, or Insert.
- **NamedPartsName** (string) – If a Named Part is requested, its name is here. If not, this is unassigned.

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

- **DynamicPartsPatterns** (PeterBlum.DES.DynamicData.DynamicPartsPatterns) – A reference to the DynamicPartsPatterns control.
- **ItemInPattern** (PeterBlum.DES.DynamicData.BaseItemInPattern) – The ItemInPattern object that is requesting this Part.
- **Host** (PeterBlum.DES.DynamicData.IDynamicPatternHost) – The DynamicPattern control.

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

GetAttributeAsNative method

Retrieves any custom Attribute declared in the ItemInPattern object. See the above description of when to use it. This method returns an object, which you probably will typecast to the desired type.

[C#]

```
object GetAttributeAsNative(string pKey)
```

[VB]

```
Function GetAttributeAsNative(ByVal pKey As String) As Object
```

Parameters

pKey

The parameter name to look for within the ItemInPattern class. This is a case sensitive value.

Returns

When the key is found in the Attributes collection, its value is returned. Otherwise it returns `null`/`Nothing`.

GetAttributeAsString method

Retrieves any custom Attribute declared in the ItemInPattern object. See the above description of when to use it. This method returns a string. If the actual value is not a string type, the `ToString()` method is called to convert to a string.

[C#]

```
string GetAttributeAsString(string pKey)
```

[VB]

```
Function GetAttributeAsString(ByVal pKey As String) As String
```

Parameters

pKey

The parameter name to look for within the ItemInPattern class. This is a case sensitive value.

Returns

When the key is found in the Attributes collection, its value is returned. Otherwise it returns `null`/`Nothing`.

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

RegisterDynamicControl method

Call from `OnPartCreating()` for each DynamicControl created. The order of these calls determines the order used by the **PositionInPattern** property of the DataFieldInPattern objects.

[C#]

```
void RegisterDynamicControl(System.Web.DynamicData.DynamicControl pDynamicControl)
```

[VB]

```
Sub RegisterDynamicControl( _
  ByVal pDynamicControl As System.Web.DynamicData.DynamicControl)
```

Parameters

pDynamicControl

The DynamicControl to register.

RegisterDynamicLabel method

Call from `OnPartCreating()` for each DynamicLabel created. The order of these calls determines the order used by the **PositionInPattern** property of the DataFieldInPattern object.

[C#]

```
void RegisterDynamicLabel(PeterBlum.DES.DynamicData.DynamicLabel pDynamicLabel)
```

[VB]

```
Sub RegisterDynamicLabel( _
  ByVal pDynamicLabel As PeterBlum.DES.DynamicData.DynamicLabel)
```

Parameters

pDynamicLabel

The DynamicLabel to register.

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

CreateFromPart method

Call to populate the container from the PartType requested. It always adds content to the end of the **Controls** collection of *pContainer*.

[C#]

```
bool CreateFromPart(Control pContainer,
  PeterBlum.DES.DynamicData.PatternPartsToGenerate pPartsToGenerate)
```

[VB]

```
Function CreateFromPart(ByVal pContainer As Control,
  ByVal pPartsToGenerate As _
    PeterBlum.DES.DynamicData.PatternPartsToGenerate ) _
    As Boolean
```

Parameters

pContainer

The control where the Template is added as child controls.

pPartsToGenerate

Same as the **PartsToGenerate** property on the DynamicPattern.

Returns

When `true`, the Part was found and applied to the Container control. When `false`, no matching Part was found and *pContainer* is unchanged.

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

Overriding the PartsPatternTemplate.OnPartCreated() method

If you want to modify the contents of a Part after its created, override the `OnPartCreated()` method. The DynamicControls and DynamicLabels have already had their **DataField** properties assigned.

Typically this is used to hide controls or modify their properties. The *Args* parameter provides a variety of details on the action, including the mode (readonly, edit, insert), and NamedPart name requested.

Your function should call the ancestor `OnPartCreated()` method to allow DES to apply the **LabelText** property found on any BaseItemInPattern subclass to the associated DynamicLabels. (If you don't want that feature, you don't have to call the ancestor.)

Here is the function definition.

[C#]

```
protected virtual void OnPartCreated(Control pContainer,
     PeterBlum.DES.DynamicData.PartCreatedEventArgs pArgs)
```

[VB]

```
Protected Overrides Sub OnPartCreated(ByVal pContainer As Control, _
     ByVal pArgs As PeterBlum.DES.DynamicData.PartCreatedEventArgs)
```

Parameters

pContainer

The control to which you find and modify controls.

To find a DynamicControl, call `pArgs.FindDynamicControl("control ID")`. When you add a DynamicLabel, call `pArgs.FindDynamicLabel("control ID")`. For any other type of control, call `pContainer.FindControl("control ID")`.

Note: It is safe to call FindControl() instead of FindDynamicControl and FindDynamicLabel. It is just a less efficient search.

pArgs

The arguments. See "The PeterBlum.DES.DynamicData.PartCreatedEventArgs class".

SEE THE NEXT PAGE FOR AN EXAMPLE

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

Example

This `OnPartCreated()` gets "DynamicLabel1" on the NamedPart named "TryMe". It sets its style sheet class to "MyClass".

[C#]

```
using DES=PeterBlum.DES.DynamicData;
...
public override void OnPartCreated(Control pContainer,
   DES.PartCreatedEventArgs pArgs)
{
   base.OnPartCreated(pContainer, pArgs);

   if ((String.Compare(pArgs.NamedPartsName, "TryMe", true) == 0)
   {
      DES.DynamicLabel vDL = pArgs.FindDynamicLabel("DynamicLabel1");
      vDL.CssClass = "MyClass";
   }
}
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
...
Public Overrides Sub OnPartCreated( _
   ByVal pContainer As Control, _
   ByVal pArgs As DES.PartCreatedEventArgs)

   MyBase.OnPartCreated(pContainer, pArgs)

   If String.Compare(pArgs.NamedPartsName, "TryMe", True) = 0 Then
      Dim vDL As DES.DynamicLabel = pArgs.FindDynamicLabel("DynamicLabel1")
      vDL.CssClass = "MyClass"
   End If
End Sub
```

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

The PeterBlum.DES.DynamicData.PartCreatedEventArgs class

This is a parameter for the `OnPartCreated()` method of the PartsPatternTemplate class. Use `OnPartCreated()` to modify controls that were generated from the requested Part.

One powerful feature of this system is that the page development can pass parameters to your `OnPartCreated()` method. Each ItemInPattern class supports "Attributes", a list of name/value pairs. The name is a string and is used to name the parameter. The value can be a string or any other type of object.

Lets suppose that you declare a Named Part called "Sample" that is implemented entirely through the `OnPartCreated()` method. You document that it accepts the parameter "ShowAll" with values of "Yes" and "No". The user can do this:

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server">
<ItemsInPattern>
  <desDD:NoDataFieldsInPattern NamedPart="Sample" ShowAll="Yes" />
</ItemsInPattern>
</desDD:DynamicPattern>
```

Your `OnPartCreated()` method will call `GetAttributeAsNative()` or `GetAttributeAsString()` to retrieve the value.

Click on any of these topics to jump to them:

- Properties of PartCreatedEventArgs
- GetAttributeAsNative method
- GetAttributeAsString method
- FindDynamicControl method
- FindDynamicLabel method
- UpdateStyleOnDynamicControls method
- UpdateStyleOnDynamicLabels method

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

Properties of PartCreatedEventArgs

The `PeterBlum.DES.DynamicData.PartCreatedEventArgs` class has these properties:

- **PartType** (enum PeterBlum.DES.DynamicData.PatternPartType) – The type of Part requested.

  The enumerated type `PeterBlum.DES.DynamicData.PatternPartType` has these values:

  - `Container` – Container around the entire control.
  - `DataRow` – Uses Mode to determine Template, EditTemplate, or Insertemplate
  - `DataCell` – Uses Mode to determine Template, EditTemplate, or Insertemplate
  - `Buttons` – Buttons within a Data Row. Used when AutoGenerateButtons is setup
  - `HeaderRow` – Header row
  - `HeaderCell` – A single field within a HeaderRow
  - `HeaderButtons` – Header for Buttons within a Header Row. Used when AutoGenerateButtons is setup
  - `FooterRow` – Footer row
  - `FooterCell` – A single field within a FooterRow
  - `FooterButtons` – Footer for Buttons within a Footer Row. Used when AutoGenerateButtons is setup
  - `DataRowSeparator` – For the ItemSeparatorTemplate of the ListView, this is a row that separates each DataRow. It can contain the entire HTML or let DataRowSeparatorCells define the appearance for each cell associated with a DataFieldInPattern object.
  - `DataRowSeparatorCell` – For the ItemSeparatorTemplate of the ListView, this a cell within a row that separates each DataRow. It only appears when DataRowSeparator indicates its use. One is generated for each DataFieldInPattern object.
  - `NoDataToShow` – When there are no rows of data to show. Generally this is a message to that face.
  - `GroupContainer` – For ListView's Grouping feature (automatic GroupTemplate generation). Defines the HTML tag that encloses a group. It should not be used for any HTML content, such as a header or footer. They are handled by GroupHeader and GroupFooter.
  - `GroupHeader` – For ListView's Grouping feature (automatic GroupTemplate generation). HTML that separates the Group's opening tag from GroupContainer from the first Item of actual data. It is often used for titles. It does not support DynamicControls.
  - `GroupFooter` – For ListView's Grouping feature (automatic GroupTemplate generation). HTML that separates the Group's closing tag from GroupContainer from the last Item of actual data. It does not support DynamicControls.
  - `GroupSeparator` – For ListView's Grouping feature (automatic GroupTemplate generation). HTML that separates groups from each other. Associated with the ListView GroupSeparatorTemplate feature.
  - `EmptyItem` – For ListView's Grouping feature. It sets up the value of the EmptyItemTemplate property. EmptyItemTemplate defines the appearance where an item would normally go in a group, but there is no data for that item.It can contain the entire HTML or let EmptyItemCells define the appearance for each cell associated with a DataFieldInPattern object.
  - `EmptyItemCell` – For the EmptyItemTemplate of the ListView, this a cell within a row that describes an record without any data. It only appears when EmptyItem indicates its use. One is generated for each DataFieldInPattern object.
  - `Unknown` – The part is not associated with the DynamicPartsPatterns, such as from a TemplateNotInPattern object.

- **Mode** (enum System.Web.UI.WebControls.DataBoundControlMode) – The mode: ReadOnly, Edit, or Insert.
- **NamedPartsName** (string) – If a Named Part is requested, its name is here. If not, this is unassigned.

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

- **DynamicPartsPatterns** (PeterBlum.DES.DynamicData.DynamicPartsPatterns) – A reference to the DynamicPartsPatterns control.
- **ItemInPattern** (PeterBlum.DES.DynamicData.BaseItemInPattern) – The ItemInPattern object that is requesting this Part.
- **Host** (PeterBlum.DES.DynamicData.IDynamicPatternHost) – The DynamicPattern control.

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

FindDynamicControl method

Retrieves a DynamicControl given its ID value.

[C#]

```
System.Web.DynamicData.DynamicControl FindDynamicControl(string pControlID)
```

[VB]

```
Function FindDynamicControl(ByVal pControlID As String) _
  As System.Web.DynamicData.DynamicControl
```

Parameters

pControlID

The value of the ID property on the DynamicControl to find.

Returns

The DynamicControl object. If you need access to properties specific to DES' DynamicControl, be sure to typecast to PeterBlum.DES.DynamicData.DynamicControl. Otherwise it returns `null`/`Nothing`.

FindDynamicLabel method

Retrieves a DynamicLabel given its ID value.

[C#]

```
PeterBlum.DES.DynamicData.DynamicLabel FindDynamicLabel(string pControlID)
```

[VB]

```
Function FindDynamicLabel(ByVal pControlID As String) _
  As PeterBlum.DES.DynamicData.DynamicLabel
```

Parameters

pControlID

The value of the ID property on the DynamicLabel to find.

Returns

The DynamicLabel object. Otherwise it returns `null`/`Nothing`.

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

UpdateStyleOnDynamicControls method

Looks through the DynamicControls that were registered. For any found, it updates its **CssClass** and **Style** properties using a **NamedStyle** and alternative **CssClass**.

[C#]

```
void UpdateStyleOnDynamicControls(string pNamedStyle, string pNamedStyleFallback,
  string pCssClass)
```

[VB]

```
Sub UpdateStyleOnDynamicControls(ByVal pNamedStyle As String, _
  ByVal pNamedStyleFallback As String, _
  ByVal pCssClass As String)
```

Parameters

pNamedStyle

The NamedStyle to apply. It searches the **NamedStyles** collection from the DynamicPattern, DynamicListView, or DynamicFormView control. Pass "" if not used.

pNamedStyleFallback

The NamedStyle to apply when *pNamedStyle* is not found. It searches the NamedStyles collection from the DynamicPattern, DynamicListView, or DynamicFormView control. Pass "" if not used.

pCssClass

The style sheet class name to apply. Pass "" if not used. Supports merging with the control's own CssClass property by using "+" before or after.

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

UpdateStyleOnDynamicLabels method

Looks through the DynamicLabel controls that were registered. For any found, it updates its **CssClass** and **Style** properties using a **NamedStyle** and alternative **CssClass**.

[C#]

```
void UpdateStyleOnDynamicLabels(string pNamedStyle, string pNamedStyleFallback,
  string pCssClass)
```

[VB]

```
Sub UpdateStyleOnDynamicLabels(ByVal pNamedStyle As String, _
  ByVal pNamedStyleFallback As String, _
  ByVal pCssClass As String)
```

Parameters

pNamedStyle

The NamedStyle to apply. It searches the **NamedStyles** collection from the DynamicPattern, DynamicListView, or DynamicFormView control. Pass "" if not used.

pNamedStyleFallback

The NamedStyle to apply when *pNamedStyle* is not found. It searches the NamedStyles collection from the DynamicPattern, DynamicListView, or DynamicFormView control. Pass "" if not used.

pCssClass

The style sheet class name to apply. Pass "" if not used. Supports merging with the control's own CssClass property by using "+" before or after.

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

Creating your PatternTemplate with the DynamicButtonsPatternTemplate class

The DynamicButtonsPatternTemplate class hosts a DynamicButtons control to provide a Pattern Template built for buttons.

While you can add a DynamicButtons control into the templates of ListView, FormView, and a PartsPatternTemplate, you may have a specific appearance for your buttons. The DynamicButtons control itself lets you provide extensive customization within its own Templates and this is where the DynamicButtonsPatternTemplate helps you most. Take a look at these Pattern Template files shipped with DES: **ButtonsForSingle.ascx**, **ButtonsForList.ascx**, and **DetailsViewButtons.ascx**. PartsPatternTemplates often point to these files.

DynamicButtonsPatternTemplate is subclassed from `PeterBlum.DES.DynamicData.NoDataFieldsPatternTemplate`. It is not intended for DynamicControls and DynamicLabels. It does not know how to use ItemInPattern objects to enhance its contents. It still supports NamedStyles.

Steps for creating a DynamicButtonsPatternTemplate

Click on any of these topics to jump to them:

- Part 1: Create the UserControl file
- Part 2: Create the controls
- Part 3: Supporting Named Styles
- Part 4: Support customization

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

Part 1: Create the UserControl file

1. Create a UserControl file. Typically its added to the **[web app]/DynamicData/PatternTemplates** folder, but it can go anywhere in your web app, so long as you set the **PatternTemplateFolderUrl** property on the DynamicPattern control.

   Suggestion: Don't use the code behind file especially if you plan on duplicating this file. Its easier to manage duplicating with a single file.

2. Modify its `<%@ Control %>` tag to inherit from `PeterBlum.DES.DynamicData.DynamicButtonsPatternTemplate`.

```
<%@ Control Language="Language option"
 Inherits="PeterBlum.DES.DynamicData.DynamicButtonsPatternTemplate" %>
```

STEPS CONTINUE ON THE NEXT PAGE

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

Part 2: Create the controls

3. Add the DynamicButtons control.

```
<desDD:DynamicButtons id="DynamicButtons1" runat="server"/>
```

4. Override the `GetDynamicButtons()` method. Its only task is to return a reference to the DynamicButtons control.

[C#]

```
protected override DynamicButtons GetDynamicButtons()
{
   return DynamicButtons1;
}
```

[VB]

```
Protected Overridden Function GetDynamicButtons() _
      As DynamicButtons
   Return DynamicButtons1
End Function
```

5. Define the formatting for the DynamicButtons control.

Often its **ReadOnlyTemplate**, **EditTemplate**, and **InsertTemplate** properties are modified to establish the order of buttons and HTML that separates them.

- The DynamicButtons Control
- Choosing the Appearance: from Properties or Templates
  - Using the Layout Properties
  - Using Templates
- Properties for the DynamicButtons Control

6. Add any desired HTML and web controls surrounding the DynamicButtons control. These elements should not depend on the mode (Readonly, edit, or insert).

7. If you want to programmatically modify properties on the DynamicButtons control prior to that control's use of the properties, override the `OnButtonsCreating()` method which is declared on the base class.

[C#]

```
protected override void OnButtonsCreating(DynamicButtons pDynamicButtons)
{
   pDynamicButtons.SaveNewButtonText = "Insert";
}
```

[VB]

```
Protected Overrides Sub OnButtonsCreating( _
    ByVal pDynamicButtons As DynamicButtons)
   pDynamicButtons.SaveNewButtonText = "Insert"
End Sub
```

STEPS CONTINUE ON THE NEXT PAGE

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

Part 3: Supporting Named Styles

For an overview of Named Styles, see "Named Styles: Your pages control the formatting".

8. Identify any HTML elements or controls that may allow the web form developer to override its styles and formatting within the DynamicPattern. What to consider:

   - Tags and controls that may get set to a style sheet class.
   - Elements that can determine the width or height. Page developers may want to adjust their height.
   - Table cells, as they can have their alignment overridden.

   If you have no need for Named Styles, you are finished at this step.

9. Define any Named Styles based on the list developed in the previous step by defining a name and description for each. It will help to include code-level documentation with this list, such as this C# code from DetailsViewButtons.ascx:

```
/* ---- USAGE -------------------------------------------------------------
    … comments about usage here …
---- NAMED STYLES -------------------------------------------------------
 Use the DynamicPattern.NamedStyles property to declare any of these style names.

   "ButtonLabelColumn" - Style for the first column (TD tag).
   "ButtonDataColumn" - Style for the second column (TD tag).
   "ButtonControl" - Style for each button.
   "ButtonRow" - Style for the TR tag.
```

10. Convert any HTML tag that will use a Named Style to a DynamicStylesTag control. Assign these properties:

    - **Tag** – The tag name.
    - **CssClass** – The class= attribute value
    - **Style** – The style= attribute value
    - For any other attributes, leave them in place as properties on the DynamicStylesTag.

    In this example, the `<td>` tag is converted.

    Original

```
<td class="ButtonLabelColumn">
  <des:Button id="EditButton" runat="server"
    CommandName="Edit" Text="Edit" CssClass="ButtonLabelControl" />
</td>
```

    Using DynamicStylesTags

```
<desDD:DynamicStylesTag runat="server" NamedStyle="ButtonLabelColumn"
  Tag="Td" CssClass="ButtonLabelColumn" >
  <des:Button id="EditButton" runat="server"
    CommandName="Edit" Text="Edit" CssClass="ButtonLabelControl" />
</desDD:DynamicStylesTag>
```

    For more detail on the DynamicStylesTag, see "The DynamicStylesTag Control".

    Note: You can use the GetTagAttributes() method in the next step or the ApplyNamedStyles() method described later to do the same work as the DynamicStylesTag.

STEPS CONTINUE ON THE NEXT PAGE

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

11. If you have an opening tag without a closing tag that needs Named Style support, declare the actual HTML tag and embed the DynamicStylesTag control within the HTML tag's header. Do not assign the **Tag** property.

```
<tr <desDD:DynamicStylesTag runat="server" NamedStyle="DataRow"
  CssClass="DataRow" > >
```

Alternatively embed a call to the `GetTagAttributes()` method (part of the PatternTemplate base class) within ASP tags like this:

```
<tr<%=GetTagAttributes("DataRow", "DataRow") %>>
```

The first parameter is the default style sheet class. It can be an empty string. The second parameter is the name of the Named Style. There are several versions of `GetTagAttributes()`. See "The PatternTemplate.GetTagAttributes method".

12. If you have a Named Style that apply to a style sheet class on individual buttons, override these two methods to return the Named Style and a default style sheet class name that is used when the Named Style is not declared in the DynamicPattern.

[C#]

```
protected override string GetButtonControlNamedStyle()
{
   return "ButtonControl";
}

protected override string GetButtonControlDefaultCssClass()
{
   return "MyButtonClass";
}
```

[VB]

```
Protected Overrides Function GetButtonControlNamedStyle() As String
   Return "ButtonControl"
End Function

Protected Overrides Function GetButtonControlDefaultCssClass() As String
   Return "MyButtonClass"
End Function
```

13. For any remaining control that needs a Named Style, call the `ApplyNamedStyle()` method from within the `OnButtonsCreated()` method. This method is declared in DynamicButtonsPatternTemplate but must be subclassed to do anything. It is passed the DynamicButtons control, whose child controls reflect the buttons, other web controls, and HTML.

You can call `pDynamicButtons.FindControl("ID")` to get a control found in a template. The ID parameter is the control's ID.

You can call `pDynamicButtons.GetButton("command name", DataBoundControlMode)` to get a button by its command name. This works for automatic generation and templates.

Here is an example that establishes the Named Styles. "ButtonContainer" effects the **CssClass** property of a DynamicButtons control.

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

Example: Apply style to the container around this control

[C#]

```
protected override void OnButtonsCreated(DynamicButtons pDynamicButtons)
{
   ApplyNamedStyle("ButtonContainer", pDynamicButtons);
}
```

[VB]

```
Protected Overrides Sub OnButtonsCreated(_
    ByVal pDynamicButtons As DynamicButtons)
   ApplyNamedStyle("ButtonContainer", pDynamicButtons)
End Sub
```

Example: Apply a style to a control within a Template

[C#]

```
protected override void OnButtonCreated(DynamicButtons pDynamicButtons)
{
   WebControl vDST =
      (WebControl) pDynamicButtons.FindControl("TRTag1");
   if (vDST != null)
      ApplyNamedStyle("ButtonRow", vDST);
}
```

[VB]

```
Protected Overrides Sub OnButtonCreated(_
    ByVal pDynamicButtons As DynamicButtons)
   Dim vDST As WebControl = _
      CType(pDynamicButtons.FindControl("TRTag1"), WebControl)
   If Not vDST Is Nothing Then
      ApplyNamedStyle("ButtonRow", vDST)
   End If
End Sub
```

Example: Apply a style to a specific button

[C#]

```
protected override void OnButtonCreated(DynamicButtons pDynamicButtons)
{
   WebControl vButton = (WebControl)
      pDynamicButtons.GetButton("Edit", DataBoundControlMode.ReadOnly);
   if (vButton != null)
      ApplyNamedStyle("Button", vButton);
}
```

[VB]

```
Protected Overrides Sub OnButtonCreated(_
    ByVal pDynamicButtons As DynamicButtons)
   Dim vButton As WebControl = _
      CType(GetButton("Edit", DataBoundControlMode.ReadOnly), WebControl)
   If Not vButton Is Nothing Then
      ApplyNamedStyle("Button", vButton)
   End If
End Sub
```

STEPS CONTINUE ON THE NEXT PAGE

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

14. *Optional.* Design mode can display the list of Named Styles in the Properties Editor if you add an XML block to your UserControl. It belongs in the C# or VB code area and must be commented out. (The PatternTemplate parser will know to ignore VB's REM statements.)

    The XML has this pattern:

```
<NamedStylesDoc>
   <NamedStyle Name="Name1">description</NamedStyle>
   <NamedStyle Name="Name2">description</NamedStyle>
      [and more]
</NamedStylesDoc>
```

    Use any of the Pattern Templates shipped with DES for examples.

STEPS CONTINUE ON THE NEXT PAGE

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

Part 4: Support customization

15. *Optional*. Create public properties that allow the user to customize the behavior of your Pattern Template. Utilize existing properties from the ancestor classes too. See "Passing properties to be used by the PatternTemplate's own code".

    When coding, override the `OnButtonsCreated()` method to change the appearance and visibility of your controls.

Example

Apply the named style in the **AltNamedStyleButtonColumn** property to the TD tag.

```
<desDD:DynamicButtons properties >
  <ReadOnlyTemplate>
    <desDD:DynamicStylesTag id="tdTag1" runat="server" Tag="Td"
         NamedStyle="ButtonLabelColumn" CssClass="ButtonLabelColumn" >
      <des:Button id="EditButton" runat="server"
           CommandName="Edit" Text="Edit" CssClass="ButtonLabelControl" />
    </desDD:DynamicStylesTag>
  </ReadOnlyTemplate>
</desDD:DynamicButtons>
```

[C#]

```
<%@ Control Language="C#" Usage="Data"
    Inherits="PeterBlum.DES.DynamicData.DynamicButtonsPatternTemplate" %>

<script runat="server">
// Supply an alternative Named Style name for the TD tag
public string AltNamedStyleButtonColumn
{
  get { return Host.GetAttributeAsBool("AltNamedStyleButtonColumn",
    cDefaultAltNamedStyleButtonColumn); }
}
private const string cDefaultAltNamedStyleButtonColumn;

protected override void OnButtonsCreated(DynamicButtons pDynamicButtons)
{
   if (!string.IsNullOrEmpty(AltNamedStyleButtonColumn))
   {
      WebControl vDST =
         (WebControl) pDynamicButtons.FindControl("tdTag1");
      if (vDST != null)
         ApplyNamedStyle(AltNamedStyleButtonColumn, vDST);
   }
}
</script>
```

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

[VB]

```
<%@ Control Language="VB" Usage="Data"
    Inherits="PeterBlum.DES.DynamicData.DynamicButtonsPatternTemplate" %>

<script runat="server">
' Supply an alternative Named Style name for the TD tag
Public ReadOnly Property AltNamedStyleButtonColumn As String
Get
    Return Host.GetAttributeAsString("AltNamedStyleButtonColumn",
      cDefaultAltNamedStyleButtonColumn)
End Get
End Property
Private Const cDefaultAltNamedStyleButtonColumn As Strin

Protected Overrides Sub OnButtonsCreated(ByVal pDynamicButtons As DynamicButtons)

   If Not String.IsNullOrEmpty(AltNamedStyleButtonColumn) Then
      Dim vDST As WebControl = _
         CType(pDynamicButtons.FindControl("tdTag1"), WebControl)
      If Not vDST Is Nothing Then
         ApplyNamedStyle(AltNamedStyleButtonColumn, vDST)
      End If
   End If
End Sub
</script>
```

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

Creating your PatternTemplate with the NoDataFieldsPatternTemplate class

The `PeterBlum.DES.DynamicData.NoDataFieldsPatternTemplate` class is used for patterns that lack DynamicControls and DynamicLabels. They may contain HTML and web controls that you want to see within a DynamicPattern, DynamicListView or DynamicFormView control, much like a UserControl. It has two capabilities beyond a UserControl:

- Supports Named Styles.
- Supports setting property values from attributes on the host control.

These templates are usually retrieved without specifying any ItemInPattern objects in the **ItemsInPattern** collection of the DynamicPattern control.

Steps for creating a NoDataFieldsPatternTemplate

Click on any of these topics to jump to them:

- Part 1: Create the UserControl file
- Part 2: Create the controls
- Part 3: Supporting Named Styles
- Part 4: Support customization

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

Part 1: Create the UserControl file

1. Create a UserControl file. Typically its added to the **[web app]/DynamicData/PatternTemplates** folder, but it can go anywhere in your web app, so long as you set the **PatternTemplateFolderUrl** property on the DynamicPattern control.

   Suggestion: Don't use the code behind file especially if you plan on duplicating this file. Its easier to manage duplicating with a single file.

2. Modify its `<%@ Control %>` tag to inherit from `PeterBlum.DES.DynamicData.NoDataFieldsPatternTemplate`.

```
<%@ Control Language="Language option"
 Inherits="PeterBlum.DES.DynamicData.NoDataFieldsPatternTemplate" %>
```

Part 2: Create the controls

3. Add any desired HTML and web controls. Do not use any DynamicControls or DynamicLabels unless you explicitly set their **DataField** property within their declaration because the Pattern Template does not retrieve them from DataFieldInPattern objects.

STEPS CONTINUE ON THE NEXT PAGE

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

Part 3: Supporting Named Styles

For an overview of Named Styles, see "Named Styles: Your pages control the formatting".

4. Identify any HTML elements or controls that may allow the web form developer to override its styles and formatting within the DynamicPattern control. What to consider:

   - Tags and controls that may get set to a style sheet class.
   - Elements that can determine the width or height. Page developers may want to adjust their height.
   - Table cells, as they can have their alignment overridden.

   If you have no need for Named Styles, you are finished at this step.

5. Define any Named Styles based on the list developed in the previous step by defining a name and description for each. It will help to include code-level documentation with this list, such as this C# code from DetailsViewButtons.ascx:

```
/* ---- USAGE ------------------------------------------------------------
    … comments about usage here …
---- NAMED STYLES -------------------------------------------------------
 Use the DynamicPattern.NamedStyles property to declare any of these style names.

   "ButtonLabelColumn" - Style for the first column (TD tag).
   "ButtonDataColumn" - Style for the second column (TD tag).
   "ButtonControl" - Style for each button.
   "ButtonRow" - Style for the TR tag.
```

6. Convert any HTML tag that will use a Named Style to a DynamicStylesTag control. Assign these properties:

   - **Tag** – The tag name.
   - **CssClass** – The class= attribute value
   - **Style** – The style= attribute value
   - For any other attributes, leave them in place as properties on the DynamicStylesTag.

   In this example, the `<td>` tag is converted.

   Original

```
<td class="ButtonLabelColumn">
  <asp:Label id="Info" runat="server" Text="Info:{0}" />
</td>
```

   Using DynamicStylesTags

```
<desDD:DynamicStylesTag runat="server" Tag="Td" NamedStyle="ButtonLabelColumn"
  CssClass="ButtonLabelColumn" >
  <asp:Label id="Info" runat="server" Text="Info:{0}" />
</desDD:DynamicStylesTag>
```

   For more detail on the DynamicStylesTag, see "The DynamicStylesTag Control".

   Note: You can use the GetTagAttributes() method in the next step or the ApplyNamedStyles() method described later to do the same work as the DynamicStylesTag.

STEPS CONTINUE ON THE NEXT PAGE

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

7. If you have an opening tag without a closing tag that needs Named Style support, declare the actual HTML tag and embed the DynamicStylesTag control within the HTML tag's header. Do not assign the **Tag** property.

```
<tr <desDD:DynamicStylesTag runat="server" NamedStyle="DataRow"
  CssClass="DataRow" > >
```

Alternatively embed a call to the `GetTagAttributes()` method (part of the PatternTemplate base class) within ASP tags like this:

```
<tr<%=GetTagAttributes("DataRow", "DataRow") %>>
```

The first parameter is the default style sheet class. It can be an empty string. The second parameter is the name of the Named Style. There are several versions of `GetTagAttributes()`. See "The PatternTemplate.GetTagAttributes method".

8. For any remaining control that needs a Named Style, call the `ApplyNamedStyle()` method from within the `Page_Load()` method.

Here is an example that establishes the Named Styles. It applies the style sheet class "InfoLabelControl" to the Label control whose ID is "Info".

Example: Apply style to the container around this control

[C#]

```
protected void Page_Load(object sender, EventArgs e)
{
   Control vInfoLabel = FindControl("Info");
   ApplyNamedStyle("InfoLabelControl", vInfoLabel);
}
```

[VB]

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
   Dim vInfoLabel As Control = FindControl("Info")
   ApplyNamedStyle("InfoLabelControl", vInfoLabel)
End Sub
```

9. *Optional.* Design mode can display the list of Named Styles in the Properties Editor if you add an XML block to your UserControl. It belongs in the C# or VB code area and must be commented out. (The PatternTemplate parser will know to ignore VB's REM statements.)

The XML has this pattern:

```
<NamedStylesDoc>
   <NamedStyle Name="Name1">description</NamedStyle>
   <NamedStyle Name="Name2">description</NamedStyle>
      [and more]
</NamedStylesDoc>
```

Use any of the Pattern Templates shipped with DES for examples.

STEPS CONTINUE ON THE NEXT PAGE

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

Part 4: Support customization

10. *Optional*. Create public properties that allow the user to customize the behavior of your Pattern Template. Utilize existing properties from the ancestor classes too. See "Passing properties to be used by the PatternTemplate's own code".

    When coding, use the `Page_Load()` method to change the appearance and visibility of your controls.

Example

Modifies the **Text** property on the label shown here to the value of the **InfoLabelText** property .

```
<desDD:DynamicStylesTag runat="server" Tag="Td" NamedStyle="ButtonLabelColumn"
  CssClass="ButtonLabelColumn" >
  <asp:Label id="Info" runat="server" Text="Info:{0}" />
</desDD:DynamicStylesTag>
```

[C#]

```
<%@ Control Language="C#" Usage="Data"
    Inherits="PeterBlum.DES.DynamicData.NoDataFieldsPatternTemplate" %>

<script runat="server">
// Supply a value for the {0} in the info label
public string InfoLabelText
{
  get { return Host.GetAttributeAsBool("InfoLabelText",
    cDefaultInfoLabelText); }
}
private const string cDefaultInfoLabelText;

protected void Page_Load(object sender, EventArgs e)
{
   Label vInfoLabel = (Label) FindControl("Info");
   if (!string.IsNullOrEmpty(InfoLabelText))
      vInfoLabel.Text = String.Format(vInfoLabel.Text, InfoLabelText);
   else
      vInfoLabel.Text = String.Format(vInfoLabel.Text, "n/a");
}

</script>
```

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

[VB]

```
<%@ Control Language="VB" Usage="Data"
    Inherits="PeterBlum.DES.DynamicData.NoDataFieldPatternTemplate" %>

<script runat="server">
' Supply a value for the {0} in the info label
Public ReadOnly Property InfoLabelText As String
Get
    Return Host.GetAttributeAsString("InfoLabelText",
      cDefaultInfoLabelText)
End Get
End Property
Private Const cDefaultInfoLabelText As String

Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
   Dim vInfoLabel As Label = CType(FindControl("Info"), Label)
   If Not String.IsNullOrEmpty(InfoLabelText) Then
      vInfoLabel.Text = String.Format(vInfoLabel.Text, InfoLabelText)
   Else
      vInfoLabel.Text = String.Format(vInfoLabel.Text, "n/a")
   End If
End Sub

</script>
```

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

Supporting Named Styles within your PatternTemplates

Click on any of these topics to jump to them:

- Overview of Named Styles
- Adding to a DataFieldsPatternTemplate
- Adding to a PartsPatternTemplate
- Adding to a DynamicButtonsPatternTemplate
- Adding to a NoDataFieldsPatternTemplate
- The PatternTemplate.GetTagAttributes method
- The PatternTemplate.ApplyNamedStyle method
- The PatternTemplate.GetCssClassFromNamedStyle method
- The PatternTemplate.GetNamedStyle method

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

The PatternTemplate.GetTagAttributes method

The `GetTagAttributes()` method is defined in the base class for Pattern Templates. Therefore it is available to all Pattern Templates.

This method supports the Named Styles feature by returning a string that can be embedded in an HTML tag. It generates the class=, style=, and a few other attributes. Its often used in markup like this:

```
<td<% =GetTagAttributes("MyClass", "MyNamedStyle") %> >
```

The string it returns replaces the `<% %>` tag and inserts a lead space character. For example:

```
<td class="MyClass" align="left">
```

It can also be used when creating HTML programmatically. For example, use a StringBuilder class to create your HTML like this within a Pattern Template:

[C#]

```
StringBuilder vHTML = new StringBuilder();
vHTML.Append("<td");
vHTML.Append(GetTagAttributes("MyClass", "MyNamedStyle"));
vHTML.Append(">");
```

[VB]

```
Dim vHTML As StringBuilder = New StringBuilder()
vHTML.Append("<td")
vHTML.Append(GetTagAttributes("MyClass", "MyNamedStyle"))
vHTML.Append(">")
```

Method declarations

[C#]

```
string GetTagAttributes(string pDefaultCssClass, string pStyleName)

string GetTagAttributes(string pDefaultCssClass, string pStyleName,
  string pFallbackStyleName)

string GetTagAttributes(string pDefaultCssClass, string pStyleName,
   string pFallbackStyleName, string pDefaultStyles)
```

[VB]

```
Function GetTagAttributes(ByVal pDefaultCssClass As String, _
  ByVal pStyleName As String) As String

Function GetTagAttributes(ByVal pDefaultCssClass As String, _
  ByVal pStyleName As String, ByVal pFallbackStyleName As String) As String

Function GetTagAttributes(ByVal pDefaultCssClass As String, _
  ByVal pStyleName As String, ByVal pFallbackStyleName As String, _
  ByVal pDefaultStyles As String)
  As String
```

Parameters

pDefaultCssClass

This is the style sheet class name used when the Named Style is not supplied in the DynamicPattern control or the Named Style does not supply its own **CssClass** property. It can be `null`/`Nothing` or "" if you don't use it. The NamedStyle can merge its own style sheet class with the DefaultCssClass by placing a + character first, like this: NamedStyle.CssClass="+myclass".

pStyleName

The name of the Named Style that will establish style and attribute formatting.

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

pFallbackStyleName

A second Named Style that is used if the DynamicPattern control does not supply the other Named Style.

pDefaultStyles

A collection of CSS style properties and values that create the style= attribute of the HTML tag. It must use the CSS syntax of

```
propertyname:value;propertyname2:value;
```

These style properties will override the same properties in the style sheet files unless they are demarked !IMPORTANT.

Returns

A string containing a lead space followed by `class="classname" style="stylename:value;stylenamevalue"`, depending on what the parameters and Named Style supplies.

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

The PatternTemplate.ApplyNamedStyle method

The `ApplyNamedStyle()` method is defined in the base class for Pattern Templates. Therefore it is available to all Pattern Templates.

This method supports the Named Styles feature by updating a control with a style sheet class name and styles from the parameters passed and a Named Style. Where its called depends on the type of Pattern Template.

DataFieldsPatternTemplate	In `Page_Load()`
PartsPatternTemplate	Override `OnPartCreated()`
DynamicButtonsPatternTemplate	Override `OnButtonsCreated()`
NoDataFieldsPatternTemplate	In `Page_Load()`

Method declarations

[C#]

```
void ApplyNamedStyle(string pStyleName, WebControl pControl)
```

```
void ApplyNamedStyle(string pStyleName, string pFallbackStyleName,
  WebControl pControl)
```

```
void ApplyNamedStyle(PeterBlum.DES.DynamicData.NamedStyle pStyle,
  WebControl pControl)
```

[VB]

```
Sub ApplyNamedStyle(ByVal pStyleName As String, ByVal pControl As WebControl)
```

```
Sub ApplyNamedStyle(ByVal pStyleName As String, ByVal pFallbackStyleName As String,_
  ByVal pControl As WebControl)
```

```
Sub ApplyNamedStyle(_
  ByVal pStyle As PeterBlum.DES.DynamicData.NamedStyle,
  ByVal pControl As WebControl)
```

Parameters

pStyleName

The name of the Named Style that will establish style and attribute formatting.

pFallbackStyleName

A second Named Style that is used if the DynamicPattern control does not supply the other Named Style.

pStyle

Use a complete NamedStyle object instead of having it looked up from the DynamicPattern control.

pControl

The control to which these styles are applied. When the control is a DynamicControl, it only supports the CssClass property. No other styles are supported.

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

Example

This is the `Page_Load()` method from the **TwoStacked.ascx** Pattern Template file.

[C#]

```
protected void Page_Load(object pSender, EventArgs pE)
{
   DynamicLabel1.Visible = ShowLabels;
   DynamicLabel1.CssClass = cCssLabelControl;
   ApplyNamedStyle("LabelControl", DynamicLabel1);

   DynamicControl1.CssClass = cCssDataControl;
   ApplyNamedStyle("DataControl", DynamicControl1);

   DynamicLabel2.Visible = ShowLabels;
   DynamicLabel2.CssClass = cCssLabelControl2;
   ApplyNamedStyle("LabelControl2", "LabelControl", DynamicLabel2);

   DynamicControl2.CssClass = cCssDataControl2;
   ApplyNamedStyle("DataControl2", "DataControl", DynamicControl2);
}
```

[VB]

```
Protected Sub Page_Load(ByVal pSender As Object, ByVal pE As EventArgs)
  DynamicLabel1.Visible = ShowLabels
  DynamicLabel1.CssClass = cCssLabelControl
  ApplyNamedStyle("LabelControl", DynamicLabel1)

  DynamicControl1.CssClass = cCssDataControl
  ApplyNamedStyle("DataControl", DynamicControl1)

  DynamicLabel2.Visible = ShowLabels
  DynamicLabel2.CssClass = cCssLabelControl2
  ApplyNamedStyle("LabelControl2", "LabelControl", DynamicLabel2)

  DynamicControl2.CssClass = cCssDataControl2
  ApplyNamedStyle("DataControl2", "DataControl", DynamicControl2)
End Sub
```

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

The PatternTemplate.GetCssClassFromNamedStyle method

The `GetCssClassFromNamedStyle()` method is defined in the base class for Pattern Templates. Therefore it is available to all Pattern Templates.

This method supports the Named Styles feature. Use it to get the style sheet class name from a Named Style. It returns a string with that class name or a default you supply if the Named Style wasn't found.

Method declarations

[C#]

```
string GetCssClassFromNamedStyle(string pStyleName, string pDefaultCssClass)
```

[VB]

```
Function GetCssClassFromNamedStyle(ByVal pStyleName As String, _
  ByVal pDefaultCssClass As String) As String
```

Parameters

pStyleName

The name of the Named Style that will establish style and attribute formatting.

pDefaultClassClass

This is the style sheet class name used when the Named Style is not supplied in the DynamicPattern control or the Named Style does not supply its own CssClass property. It can be `null`/`Nothing` or "" if you don't use it.

Returns

The style sheet class name found, or "" if not found or the Named Style did not have its **CssClass** property assigned.

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

The PatternTemplate.GetNamedStyle method

The `GetNamedStyle()` method is defined in the base class for Pattern Templates. Therefore it is available to all Pattern Templates.

This method supports the Named Styles feature. Use it to get the NamedStyle object for the requested Named Style.

Method declarations

[C#]

```
PeterBlum.DES.DynamicData.NamedStyle GetNamedStyle(
   string pStyleName)
```

```
PeterBlum.DES.DynamicData.NamedStyle GetNamedStyle(
   string pStyleName, string pDataField)
```

[VB]

```
Function GetNamedStyle(ByVal pStyleName As String) _
   As PeterBlum.DES.DynamicData.NamedStyle
```

```
Function GetNamedStyle(ByVal pStyleName As String, _
   ByVal pDataField As String) _
   As PeterBlum.DES.DynamicData.NamedStyle
```

Parameters

pStyleName

The name of the Named Style that will establish style and attribute formatting.

pDataField

If the NamedStyle has its **DataField** property assigned, use this to locate that object by both its Named Style and data field name.

Returns

The NamedStyle object or `null`/`Nothing` if not found.

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

The DynamicPartsPatterns Control

The DynamicPartsPatterns control must be used within a PartsPatternTemplate, which is a Pattern Template file subclassed from `PeterBlum.DES.DynamicData.PartsPatternTemplate`. Consider these two a matched pair: one always goes with the other.

The DynamicPartsPatterns control provides a number of "Parts" containing HTML and web controls. Some are expected to contain DynamicControls that get their **DataField** property assigned from a DynamicPattern, DynamicListView, or DynamicFormView control. Some are used to show column titles. Some are used to show buttons. Each part can offer a list of alternatives for their output, called "NamedParts". That way you can use either the standard or Named Part to allow your DynamicPattern, DynamicListView, and DynamicFormView controls to build the right interface for the situation.

All combined, these generates the parts of a list-style and single record style DataBound control's user interface. The **GridView.ascx** , **ListOfDetailViews.ascx**, and **DetailsView.ascx** Pattern Template files use the DynamicPartsPatterns control.

Click on any of these topics to jump to them:

- Parts Overview
- NamedParts Overview
- Properties of the DynamicPartsPatterns Control
- Properties on each Part


```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <DataCell NamedStyle="DataCell" CssClass="DataCellStyle"
    Style="vertical-align:top;" >
    <Template>
      <desDD:DynamicControl ID="DynamicControl1" runat="server" />
    </Template>
  </DataCell>
  <NoDataToShow Tag="Div" NamedStyle="NoDataToShow"
    CssClass="NoDataToShowStyle" Message="No records were found." >
  </NoDataToShow>
</desDD:DynamicPartsPatterns>
```

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

Parts Overview

The DynamicPartsPatterns control provides a list of Parts properties. Each Part describes one element of a DataBound control, such as the container tag around the entire control, the header row, data rows, and what to show when there is no data available. It has Parts for special cases of the ListView and DynamicListView controls, such as for Grouping.

Since Parts are properties of DynamicPartsPattern, the ASP.NET markup declares them as XML nodes like this:

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <PartName properties that determine the containing HTML tag >
    <Template>
      Controls and HTML to output
    </Template>
  </PartName>
</desDD:DynamicPartsPatterns>
```

Here are the Part properties available

Property Name	Description
Container	The HTML tag around the entire control. Specify its **Tag** property to define the HTML tag. It does not supply a **Template** property for any of its inner content because the rest of the tags are the inner content.
DataRow	Describes the content for a single record. This Part tag provides three templates that define different output based on the data entry mode. You can declare the DynamicControls here, but if you want them to be generated for each data field automatically, use the **DataCell** for that and use **DataRow** to define the HTML tag that encloses the list of **DataCell** parts.
DataCell	Describes the content of a single field in the record. This Part tag provides three templates that define different output based on the data entry mode. It should have at least one DynamicControl. It is used when there is no Template defined in **DataRow** or if the **DataRow.Template** includes a PlaceHolder for the **DataCell** parts.
Buttons	Describes the element that shows buttons for the record. The buttons depend on the data entry mode, such as Edit and Details Buttons for ReadOnly mode and Save and Cancel Buttons for Insert and Edit modes. This part often contains a DynamicButtons control which knows the correct buttons to show based on the mode and various other settings in your form. This part is only used when the **DataRow** also allows the output of **DataCells**. It depends on either the **AutoGenerateButtons** property or the presence of a ButtonsInPattern object in the **ItemsInPattern** collection.
HeaderRow	Describes the content for a header. It often includes column titles. If your titles come from the data field names, you can declare the DynamicColumnTitle controls here, but if you want them to be generated for each data field automatically, use the **HeaderCell** for that and use **HeaderRow** to define the HTML tag that encloses the list of **HeaderCell** parts.
HeaderCell	Describes the content of a single field's header, such as a column title. It usually has a DynamicColumnTitle or DynamicLabel which will automatically be assigned the data field name. It is used when there is no Template defined in **HeaderRow** or if the **HeaderRow.Template** includes a PlaceHolder for the **HeaderCell** parts.
HeaderButtons	Describes a header for a column containing Buttons parts. It is often the column title for buttons. If it does not need a title, it often uses `" "`. It is only used when **HeaderCell** is used and the **AutoGenerateButtons** property is not set to `None`.

CONTINUED ON THE NEXT PAGE

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

Property Name	Description
FooterRow	Describes the content for a Footer after to the list of records. If you want one element generated for each data field, use the **FooterCell** part.
FooterCell	Describes the content of a single field's Footer. It is used when there is no Template defined in **FooterRow** or if the **FooterRow.Template** includes a PlaceHolder for the **FooterCell** parts.
FooterButtons	Describes a Footer for a column containing **Buttons** parts. It is only used when **FooterCell** is used and the **AutoGenerateButtons** property is not set to `None`.
NoDataToShow	Used when there is no data found. It is often a message like "No data found."

The following properties are specific to List-style interfaces and are generally used with the ListView and DynamicListView controls.

Property Name	Description
DataRowSeparator	A row that separates each record. If you want one element generated for each data field, use the **DataRowSeparatorCell** part. Associated with the **ListView.ItemSeparatorTemplate**.
DataRowSeparatorCell	A single field's content for the **DataRowSeparator**. It is used when there is no Template defined in **DataRowSeparator** or if the **DataRowSeparator.Template** includes a PlaceHolder for the **DataRowSeparatorCell** parts.
GroupContainer	Associated with ListView.**GroupTemplate**. **GroupContainer** is the HTML tag that encloses a group. Specify its **Tag** property to define the HTML tag. It does not supply a **Template** property for any of its inner content because the rest of the tags are the inner content.
GroupHeader	The header for the grouping feature. HTML that separates the **GroupContainer's** opening tag from the first Item of actual data.
GroupFooter	The footer for the grouping feature. HTML that separates the last Item of actual data from the **GroupContainer**'s closing tag.
GroupSeparator	HTML that separates groups from each other. Associated with the **ListView.GroupSeparatorTemplate**.
EmptyItem	Defines the appearance where an item would normally go in a group, but there is no data for that item. It can contain the entire HTML or let **EmptyItemCells** define the appearance for each cell associated with a DataFieldInPattern object. Associated with **ListView.EditItemTemplate**.
EmptyItemCell	This a cell within a row that describes an record without any data. It only appears when **EmptyItem** indicates its use. One is generated for each DataFieldInPattern object.

Selecting the Parts from the DynamicPattern control

The DynamicListView and DynamicFormView controls know exactly how to utilize the Parts from the PartsPatternTemplate. If you are using the DynamicPattern control, you must explicitly request one or more Parts through its **PartsToGenerate** property.

See "Working with a PartsPatternTemplate".

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="GridView" PartsToGenerate="HeaderRow" />
```

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
   PatternTemplateName="GridView" PartsToGenerate="DataRow" Mode="Edit" />
```

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

NamedParts Overview

Each Part property provides its default HTML container tag and inner contents unless you specify an alternative by its name from the **NamedParts** collection of the Parts. NamedParts also define the container tag and inner contents. They allow you to define many appearances for the Part.

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <PartName properties that determine the containing HTML tag >
    <Template>
      Controls and HTML to output
    </Template>
    <NamedParts>
      <desDD:PartTypeNamedPatternPart Name="name" UseStandardContainer="boolean"
        properties that determine the containing HTML tag
        when UseStandardContainer="false" >
        <Template>
          Controls and HTML to output
        </Template>
      </desDD:PartTypeNamedPatternPart>
    </NamedParts>
  </PartName>
</desDD:DynamicPartsPatterns>
```

Always define a value in the **Name** property. It must be unique amongst all other NamedParts in the same Part, but it can be identical to a NamedPart in other Parts.

The NamePart will reuse inner content of the standard Part unless you setup its inner content properties like **Template** and **PatternTemplateName**.

The NamedPart will reuse the container tag properties of the standard Part unless you set its **UseStandardContainer** property to `false`. It defaults to `true`. Only then will its own **CssClass**, **Style**, **ElementAttributes**, **NamedStyle**, and **NamedStyleFallback** will be used to generate the container tag.

Examples of the UseStandardContainer property

The "SelectButton" NamedPart will use "Button", not "AltButton" for its style sheet class. It will use "ButtonControl", not "AltButtonControl" for the NamedStyle.

```
<Buttons CssClass="Button" NamedStyle="ButtonControl" >
  <NamedParts>
    <desDD:ButtonsNamedPatternPart Name="SelectButton"
      CssClass="AltButton" NamedStyle="AltButtonControl"  >
    </desDD:ButtonsNamedPatternPart>
  </NamedParts>
</Buttons>
```

Because **UseStandardContainer** is `false`, the "SelectButton" NamedPart will use "AltButton" for its style sheet class. It will use "AltButtonControl" for the NamedStyle.

```
<Buttons CssClass="Button" NamedStyle="ButtonControl" >
  <NamedParts>
    <desDD:ButtonsNamedPatternPart Name="SelectButton"
      UseStandardContainer="false"
      CssClass="AltButton" NamedStyle="AltButtonControl"  >
    </desDD:ButtonsNamedPatternPart>
  </NamedParts>
</Buttons>
```

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

Using NamedParts from DynamicListView, DynamicFormView and DynamicPattern controls

When you want to use a NamedPart, you can define it for all cases requested by your DynamicListView, DynamicFormView, or DynamicPattern control on the ***PartType*NamedPart** property (such as ButtonsNamedPart for the Buttons Part.) See "Default NamedParts Properties" for a list of these properties.

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
  PatternTemplateName="GridView" PartsToGenerate="DataRow"
  ButtonsNamedPart="Buttons" />
```

Each most ItemInPattern objects offer a **NamedPart** property where you can specify a name of a NamedPart. This provides an alternative on a case-by-case basis. For some ItemInPattern objects, the NamedPart is typically used. For example, the DataFieldListInPattern object picks the NamedPart and supplies a list of data fields to apply when there are multiple DynamicControls within your Named Part.

See "A NamedPart with more than one data field".

```
<desDD:DynamicPattern id="DynamicPattern1" runat="server"
  PatternTemplateName="GridView" PartsToGenerate="DataCell" >
  <ItemsInPattern>
    <desDD:DataFieldListInPattern NamedPart="TwoStacked" />
      <ItemsInPattern>
        <desDD:DataFieldInPattern DataField="columnname1" />
        <desDD:DataFieldInPattern DataField="columnname2" />
      </ItemsInPattern>
    </desDD:DataFieldListInPattern>
  </ItemsInPattern>
</desDD:DynamicPattern>
```

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

Properties of the DynamicPartsPatterns Control

DynamicPartsPatterns is subclassed from `System.Web.UI.Control`. Aside from the standard properties, it introduces a number of templates whose contents are used by the DynamicPattern control to generate elements of a data record, including header, footer, and the record data itself.

Please see the **GridView.ascx**, **ListOfDetailViews.ascx**, and **DetailsView.ascx** Pattern Template files for examples.

Properties by Part
Click on any of these topics to jump to them:

- Using the Container Part
- Using the DataRow Part
- Using the DataCell Part
- Using the Buttons Part
- Using the HeaderRow Part
- Using the HeaderCell Part
- Using the HeaderButtons Part
- Using the FooterRow Part
- Using the FooterCell Part
- Using the FooterButtons Part
- Using the NoDataToShow Part
- Using the DataRowSeparator Part
- Using the DataRowSeparatorCell Part
- Using the GroupContainer Part
- Using the GroupHeader Part
- Using the GroupFooter Part
- Using the GroupSeparator Part
- Using the EmptyItem Part
- Using the EmptyItemCell Part

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

Using the Container Part

NamedPart class	ContainerNamedPatternPart
Use in ListView within	LayoutTemplate
Use in FormView within	HeaderTemplate, FooterTemplate
Available Templates	None
Binds data fields to	None
When using the DynamicPattern Control:	
PartsToGenerate values	ContainerOpeningTag, ContainerClosingTag, DetailsView
Other DynamicPattern properties	None

The HTML tag around the entire control. Specify its **Tag** property to define the HTML tag. It does not supply a **Template** property for any of its inner content because the rest of the tags are the inner content.

When you set **Tag** to `Table`, the Rows and cells will immediately use `<tr>` and `<td>` or `<th>` tags because all **Tag** properties default to `Auto`. Similarly if you set **Tag** to `Ol` or `Ul`, the Rows will immediately use the `<li>` tag.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <Container Tag="value" CssClass="value" Style="value"
    NamedStyle="value" NamedStyleFallback="value" >
    <NamedParts>
      <desDD:ContainerNamedPatternPart Name="name" UseStandardContainer="boolean"
        Same properties as <Container> >
      </desDD:ContainerNamedPatternPart>
    </NamedParts>
  </Container>
</desDD:DynamicPartsPatterns>
```

Properties on this Part
Click on any of these topics to jump to them:

- Container Tag Properties
- NamedParts Properties
- Behavior Properties

Using this Part within a ListView control

On a ListView, use it in the **LayoutTemplate**. Define two, one for the opening containing tag before everything else and another for the closing containing tag after. When using the ListView's grouping feature, switch from LayoutTemplate to **GroupTemplate**.

```
<LayoutTemplate>
  <desDD:DynamicPattern id="DPOpening" runat="server"
    PatternTemplateName="name" PartsToGenerate="ContainerOpeningTag" />

  HTML and controls to generate the headers, data rows and footers

  <desDD:DynamicPattern id="DPClosing" runat="server"
    PatternTemplateName="name" PartsToGenerate="ContainerClosingTag" />
</LayoutTemplate>
```

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

Using this Part within a FormView control

On a FormView, define the opening containing tag in the **HeaderTemplate** and the closing containing tag in the **FooterTemplate**.

```
<HeaderTemplate>
  <desDD:DynamicPattern id="DPOpening" runat="server"
    PatternTemplateName="name" PartsToGenerate="ContainerOpeningTag" />

  HTML and controls to generate the header

</HeaderTemplate>

<FooterTemplate>
  HTML and controls to generate the footer

  <desDD:DynamicPattern id="DPClosing" runat="server"
    PatternTemplateName="name" PartsToGenerate="ContainerClosingTag" />
</FooterTemplate>
```

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

Using the DataRow Part

NamedPart class	DataRowNamedPatternPart
Use in ListView within	ItemTemplate, EditItemTemplate, InsertItemTemplate, SelectedItemTemplate, AlternatingItemTemplate
Use in FormView within	ItemTemplate, EditItemTemplate, InsertItemTemplate
Available Templates	Template, EditTemplate, InsertTemplate
Binds data fields to	DynamicControl, DynamicLabel
When using the DynamicPattern Control:	
PartsToGenerate values	DataRow, DataRowReadOnly, DataRowEdit, DataRowInsert, DetailsView
Other DynamicPattern properties	Mode, DataRowAppearance

Describes the content for a single record.

There are two ways to output data fields in DynamicControls.

- Define the complete HTML layout with DynamicControls within the DataRow. Do not use the DataCells.

  Use this when you have a layout that varies its appearance depending on the data field and you know exactly how many data fields will be offered. You will add exactly that number of DynamicControls.

- Define the inner content using the DataCell Part. Use the DataRow to establish a row header and footer than encloses all of the DataCell Parts. It determines the number of DataCells from the list of DataFieldInPattern objects within the **ItemsInPattern** collection on the DynamicPattern, DynamicListView, or DynamicFormView controls.

  Use this when all data fields will appear with identical HTML around their DynamicControl, the way the GridView control builds table cells around data fields for columns and DetailsView control builds a table row around a DynamicLabel and DynamicColumn control.

  If you define any content in the **Template**, **EditTemplate**, or **InsertTemplate** properties and intend to use the DataCell, add a PlaceHolder control where a list of DataCells will be inserted and set the **CellPlaceHolderID** property to the ID of that PlaceHolder control.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
   <DataRow Tag="value" CssClass="value" Style="value"
      NamedStyle="value" NamedStyleFallback="value" CellPlaceHolderID="ID"
      SelectedCssClass="value" SelectedNamedStyle="value"
      SelectedNameStyleFallback="value"
      AlternativeCssClass="value" AlternativeNamedStyle="value"
      AlternativeNameStyleFallback="value"
      PatternTemplateName="name" EditPatternTemplateName="name"
      InsertPatternTemplateName="name"  >
      <Template>
         HTML, DynamicControls, and other controls
         If using DataCells, include a PlaceHolder control where they will go
      </Template>
      <EditTemplate />
      <InserTemplate />
      <NamedParts>
         <desDD:DataRowNamedPatternPart Name="name" UseStandardContainer="boolean"
            Same properties as <DataRow> >
            <Template />
            <EditTemplate />
            <InserTemplate />
         </desDD:DataRowNamedPatternPart>
      </NamedParts>
   </DataRow>
</desDD:DynamicPartsPatterns>
```

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

Properties on this Part
Click on any of these topics to jump to them:

- Container Tag Properties
- Inner content Properties
- NamedParts Properties
- Behavior Properties

Example: DynamicControls within DataRow; DataCells not used

This supports 3 data fields. The first two appear side-by-side. The third goes under the other two, separated by a solid line.

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <DataRow Tag="Div" CssClass="DataRow" NamedStyle="DataRow"
    SelectedCssClass="SelectedDataRow" SelectedNamedStyle="SelectedDataRow"
    AlternativeCssClass="AltDataRow" AlternativeNamedStyle="AltDataRow" >
    <Template>
      <desDD:DynamicLabel id="FieldLabel1" runat="server"
           AssociatedControlID="FieldData1" />: 
      <desDD:DynamicControl id="FieldData1" runat="server"/>:  
      <desDD:DynamicLabel id="FieldLabel2" runat="server"
           AssociatedControlID="FieldData2" />: 
      <desDD:DynamicControl id="FieldData2" runat="server"/><br />
      <hr />
      <desDD:DynamicLabel id="FieldLabel3" runat="server"
           AssociatedControlID="FieldData3" />: 
      <desDD:DynamicControl id="FieldData3" runat="server" /><br />
    </Template>
  </DataRow>
</desDD:DynamicPartsPatterns>
```

Example: DataCells for DynamicControls; DataRow provides the Table Row tag

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <DataRow Tag="Tr" CssClass="DataRow" NamedStyle="DataRow"
    SelectedCssClass="SelectedDataRow" SelectedNamedStyle="SelectedDataRow"
    AlternativeCssClass="AltDataRow" AlternativeNamedStyle="AltDataRow" >
  </DataRow>
  <DataCell Tag="Td" CssClass="DataCell" NamedStyle="DataCell" >
    <Template>
      <desDD:DynamicControl id="FieldData1" runat="server"/>
    </Template>
  </DataCell>
</desDD:DynamicPartsPatterns>
```

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

Example: DataCells for DynamicControls; DataRow provides a header

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <DataRow Tag="Div" CssClass="DataRow" NamedStyle="DataRow"
    SelectedCssClass="SelectedDataRow" SelectedNamedStyle="SelectedDataRow"
    AlternativeCssClass="AltDataRow" AlternativeNamedStyle="AltDataRow"
    CellPlaceHolderID="DataCellPlaceHolder" >
    <Template>
      <hr />  <!-- separator before the DataCells -->
      <asp:PlaceHolder id="DataCellPlaceHolder" runat="server" />
    </Template>
  </DataRow>
  <DataCell Tag="Div" CssClass="DataCell" NamedStyle="DataCell" >
    <Template>
      <desDD:DynamicControl id="FieldData1" runat="server"/>
    </Template>
  </DataCell>
</desDD:DynamicPartsPatterns>
```

Using this Part within a ListView control

On a ListView, use it in all of the ItemTemplates. **EditItemTemplate** and **InsertItemTemplate** use the **Mode** property. **SelectedItemTemplate** and **AlternatingItemTemplate** use the **DataRowAppearance** property.

```
<ItemTemplate>
  <desDD:DynamicPattern id="DPReadOnly" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataRow"
    AutoGenerateButtons="First" />
</ItemTemplate>

<EditItemTemplate>
  <desDD:DynamicPattern id="DPEdit" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataRow" Mode="Edit"
    AutoGenerateButtons="First" />
</EditItemTemplate>

<InsertItemTemplate>
  <desDD:DynamicPattern id="DPInsert" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataRow" Mode="Insert"
    AutoGenerateButtons="First" />
</InsertItemTemplate>

<SelectedItemTemplate>
  <desDD:DynamicPattern id="DPSelected" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataRow"
    DataRowAppearance="Selected" AutoGenerateButtons="First" />
</SelectedItemTemplate>

<AlternatingItemTemplate>
  <desDD:DynamicPattern id="DPSelected" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataRow"
    DataRowAppearance="Alternative" AutoGenerateButtons="First" />
</AlternatingItemTemplate>
```

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

Using this Part within a FormView control

On a FormView, use it in all of the ItemTemplates. **EditItemTemplate** and **InsertItemTemplate** use the **Mode** property.

```
<ItemTemplate>
  <desDD:DynamicPattern id="DPReadOnly" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataRow" />
    AutoGenerateButtons="First" />
</ItemTemplate>

<EditItemTemplate>
  <desDD:DynamicPattern id="DPEdit" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataRow" Mode="Edit"
    AutoGenerateButtons="First" />
</EditItemTemplate>

<InsertItemTemplate>
  <desDD:DynamicPattern id="DPInsert" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataRow" Mode="Insert"
    AutoGenerateButtons="First" />
</InsertItemTemplate>
```

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

Using the DataCell Part

NamedPart class	DataCellNamedPatternPart
Use in ListView within	Generally used indirectly through the DataRow Part
Use in FormView within	Generally used indirectly through the DataRow Part
Available Templates	Template, EditTemplate, InsertTemplate
Binds data fields to	DynamicControl, DynamicLabel
When using the DynamicPattern Control:	
PartsToGenerate values	DataCell, DataCellReadOnly, DataCellEdit, DataCellInsert, DetailsView
Other DynamicPattern properties	Mode, DataRowAppearance

Describes the content for a single field withing a record. When not setup, it generates a DynamicControl as its content.

There are two ways to output data fields in DynamicControls.

- Define the complete HTML layout with DynamicControls within the DataRow. Do not use the DataCells.

  Use this when you have a layout that varies its appearance depending on the data field and you know exactly how many data fields will be offered. You will add exactly that number of DynamicControls.

- Define the inner content using the DataCell Part. Use the DataRow to establish a row header and footer than encloses all of the DataCell Parts. It determines the number of DataCells from the list of DataFieldInPattern objects within the **ItemsInPattern** collection on the DynamicPattern, DynamicListView, or DynamicFormView controls.

  Use this when all data fields will appear with identical HTML around their DynamicControl, the way the GridView control builds table cells around data fields for columns and DetailsView control builds a table row around a DynamicLabel and DynamicColumn control.

  If you define any content in the DataRow's **Template**, **EditTemplate**, or **InsertTemplate** properties, also add a PlaceHolder control where a list of DataCells will be inserted and set the DataRow's **CellPlaceHolderID** property to the ID of that PlaceHolder control.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <DataCell Tag="value" CssClass="value" Style="value"
    NamedStyle="value" NamedStyleFallback="value"
    SelectedCssClass="value" SelectedNamedStyle="value"
    SelectedNameStyleFallback="value"
    AlternativeCssClass="value" AlternativeNamedStyle="value"
    AlternativeNameStyleFallback="value"
    PatternTemplateName="name" EditPatternTemplateName="name"
    InsertPatternTemplateName="name"  >
    <Template>
      HTML, DynamicControls, and other controls
    </Template>
    <EditTemplate />
    <InserTemplate />
    <NamedParts>
      <desDD:DataCellNamedPatternPart Name="name" UseStandardContainer="boolean"
        Same properties as <DataCell> >
        <Template />
        <EditTemplate />
        <InserTemplate />
      </desDD:DataCellNamedPatternPart>
    </NamedParts>
  </DataCell>
</desDD:DynamicPartsPatterns>
```

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

Properties on this Part
Click on any of these topics to jump to them:

- Container Tag Properties
- Inner content Properties
- NamedParts Properties
- Behavior Properties

Example: DataCells for DynamicControls; DataRow provides the Table Row tag

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <DataRow Tag="Tr" CssClass="DataRow" NamedStyle="DataRow"
    SelectedCssClass="SelectedDataRow" SelectedNamedStyle="SelectedDataRow"
    AlternativeCssClass="AltDataRow" AlternativeNamedStyle="AltDataRow" >
  </DataRow>
  <DataCell Tag="Td" CssClass="DataCell" NamedStyle="DataCell" >
    <Template>
      <desDD:DynamicControl id="FieldData1" runat="server"/>
    </Template>
  </DataCell>
</desDD:DynamicPartsPatterns>
```

Example: DataCells for DynamicControls; DataRow provides a header

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <DataRow Tag="Div" CssClass="DataRow" NamedStyle="DataRow"
    SelectedCssClass="SelectedDataRow" SelectedNamedStyle="SelectedDataRow"
    AlternativeCssClass="AltDataRow" AlternativeNamedStyle="AltDataRow"
    CellPlaceHolderID="DataCellPlaceHolder" >
    <Template>
      <hr />  <!-- separator before the DataCells -->
      <asp:PlaceHolder id="DataCellPlaceHolder" runat="server" />
    </Template>
  </DataRow>
  <DataCell Tag="Div" CssClass="DataCell" NamedStyle="DataCell" >
    <Template>
      <desDD:DynamicControl id="FieldData1" runat="server"/>
    </Template>
  </DataCell>
</desDD:DynamicPartsPatterns>
```

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

Using this Part within a ListView control

Generally you will use the DataRow, not the DataCell, from within a ListView control. However, you can use DataCell so long as you supply the **DataField** property.

On a ListView, use it in all of the ItemTemplates. **EditItemTemplate** and **InsertItemTemplate** use the **Mode** property. **SelectedItemTemplate** and **AlternatingItemTemplate** use the **DataRowAppearance** property.

Only the **ItemTemplate** is shown here.

```
<ItemTemplate>
  <frameset class="RecordEnclosure">
    <legend>
      <desDD:DynamicControl id="DPProductName" runat="server"
        DataField="ProductName" />
    </legend>
  <desDD:DynamicPattern id="DPUnitPrice" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataCell" DataField="UnitPrice" />
  <br />
  <desDD:DynamicPattern id="DPDescription" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataCell" DataField="Description" />
  <desDD:DynamicPattern id="DPButtons" runat="server"
    PatternTemplateName="name" PartsToGenerate="Buttons" />
  </frameset>
</ItemTemplate>
```

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

Using this Part within a FormView control

Generally you will use the DataRow, not the DataCell, from within a FormView control. However, you can use DataCell so long as you supply the **DataField** property.

On a FormView, use it in all of the ItemTemplates. **EditItemTemplate** and **InsertItemTemplate** use the **Mode** property.

Only the **ItemTemplate** is shown here.

```
<ItemTemplate>
  <frameset class="RecordEnclosure">
    <legend>
      <desDD:DynamicControl id="DPProductName" runat="server"
        DataField="ProductName" />
    </legend>
  <desDD:DynamicPattern id="DPUnitPrice" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataCell" DataField="UnitPrice" />
  <br />
  <desDD:DynamicPattern id="DPDescription" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataCell" DataField="Description" />
  <desDD:DynamicPattern id="DPButtons" runat="server"
    PatternTemplateName="name" PartsToGenerate="Buttons" />
  </frameset>
</ItemTemplate>
```

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

Using the Buttons Part

NamedPart class	ButtonsNamedPatternPart
Use in ListView within	Generally used indirectly through the DataRow Part
Use in FormView within	Generally used indirectly through the DataRow Part
Available Templates	Template, EditTemplate, InsertTemplate
Binds data fields to	None
When using the DynamicPattern Control:	
PartsToGenerate values	Buttons, DetailsView
Other DynamicPattern properties	Mode, DataRowAppearance

Describes the element that shows buttons for the record. Generally this is a DynamicButtons control, which is the default if you do not define anything in the **Template** or **PatternTemplateName** properties.

This part is only used when the **DataRow** also allows the output of **DataCells**. It depends on either the **AutoGenerateButtons** property on DynamicPattern, DynamicListView, or DynamicFormView controls or on the presence of a ButtonsInPattern object in the **ItemsInPattern** collection.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <Buttons Tag="value" CssClass="value" Style="value"
    NamedStyle="value" NamedStyleFallback="value"
    SelectedCssClass="value" SelectedNamedStyle="value"
    SelectedNameStyleFallback="value"
    AlternativeCssClass="value" AlternativeNamedStyle="value"
    AlternativeNameStyleFallback="value"
    PatternTemplateName="name" EditPatternTemplateName="name"
    InsertPatternTemplateName="name"  >
    <Template>
      HTML, DynamicButtons control, and other controls
    </Template>
    <EditTemplate />
    <InserTemplate />
    <NamedParts>
      <desDD:ButtonsNamedPatternPart Name="name" UseStandardContainer="boolean"
        Same properties as <Buttons> >
        <Template />
        <EditTemplate />
        <InserTemplate />
      </desDD:ButtonsNamedPatternPart>
    </NamedParts>
  </Buttons>
</desDD:DynamicPartsPatterns>
```

Click on any of these topics to jump to them:

- Container Tag Properties
- Inner content Properties
- NamedParts Properties
- Behavior Properties

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

Example

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <DataRow Tag="Tr" CssClass="DataRow" NamedStyle="DataRow"
    SelectedCssClass="SelectedDataRow" SelectedNamedStyle="SelectedDataRow"
    AlternativeCssClass="AltDataRow" AlternativeNamedStyle="AltDataRow" >
  </DataRow>
  <DataCell Tag="Td" CssClass="DataCell" NamedStyle="DataCell" >
    <Template>
      <desDD:DynamicControl id="FieldData1" runat="server"/>
    </Template>
  </DataCell>
  <Buttons Tag="Td" CssClass="Buttons" NamedStyle="Buttons" >
    <Template>
      <desDD:DynamicButtons id="Buttons1" runat="server"/>
    </Template>
  </Buttons>
</desDD:DynamicPartsPatterns>
```

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

Using this Part within a ListView control

Generally you will use the DataRow Part, not the Buttons Part, from within a ListView control. The DynamicPattern control will use the **AutoGenerateButtons** property or a ButtonsInPattern object in the **ItemsInPattern** property to request the Buttons Part. However, you can use the Buttons part as shown here.

On a ListView, use it in all of the ItemTemplates. **EditItemTemplate** and **InsertItemTemplate** use the **Mode** property. **SelectedItemTemplate** and **AlternatingItemTemplate** use the **DataRowAppearance** property.

Only the **ItemTemplate** is shown here.

```
<ItemTemplate>
  <frameset class="RecordEnclosure">
    <legend>
      <desDD:DynamicControl id="DPProductName" runat="server"
        DataField="ProductName" />
    </legend>
  <desDD:DynamicPattern id="DPUnitPrice" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataCell" DataField="UnitPrice" />
  <br />
  <desDD:DynamicPattern id="DPDescription" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataCell" DataField="Description" />
  <desDD:DynamicPattern id="DPButtons" runat="server"
    PatternTemplateName="name" PartsToGenerate="Buttons" />
  </frameset>
</ItemTemplate>
```

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

Using this Part within a FormView control

Generally you will use the DataRow Part, not the Buttons Part, from within a FormView control. The DynamicPattern control will use the **AutoGenerateButtons** property or a ButtonsInPattern object in the **ItemsInPattern** property to request the Buttons Part from the DataRow Part. However, you can use the Buttons part as shown here.

On a FormView, use it in all of the ItemTemplates. **EditItemTemplate** and **InsertItemTemplate** use the **Mode** property.

Only the **ItemTemplate** is shown here.

```
<ItemTemplate>
  <frameset class="RecordEnclosure">
    <legend>
      <desDD:DynamicControl id="DPProductName" runat="server"
        DataField="ProductName" />
    </legend>
  <desDD:DynamicPattern id="DPUnitPrice" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataCell" DataField="UnitPrice" />
  <br />
  <desDD:DynamicPattern id="DPDescription" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataCell" DataField="Description" />
  </frameset>
  <desDD:DynamicPattern id="DPButtons" runat="server"
    PatternTemplateName="name" PartsToGenerate="Buttons" />
</ItemTemplate>
```

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

Using the HeaderRow Part

NamedPart class	HeaderRowNamedPatternPart
Use in ListView within	HeaderTemplate
Use in FormView within	HeaderTemplate
Available Templates	Template
Binds data fields to	DynamicColumnTitles, DynamicLabel
When using the DynamicPattern Control:	
PartsToGenerate values	HeaderRow, DetailsView
Other DynamicPattern properties	AutoGenerateButtons

Describes the content for a header. It often includes column titles. The text for these labels comes from either the **DataField** property on a DataFieldInPattern object or the **HeaderText** property on the DataFieldInPattern or LabelInPattern objects.

There are two ways to output data fields in DynamicLabels or DynamicColumnTitles.

- Define the complete HTML layout with DynamicLabels within the HeaderRow. Do not use the HeaderCells.

  Use this when you have a layout that varies its appearance depending on the data field and you know exactly how many data fields will be offered. You will add exactly that number of DynamicLabels.

- Define the inner content using the HeaderCell Part. Use the HeaderRow to establish a row header and footer than encloses all of the HeaderCell Parts. It determines the number of HeaderCells from the list of DataFieldInPattern objects within the **ItemsInPattern** collection on the DynamicPattern, DynamicListView, or DynamicFormView controls.

  Use this when all data fields titles will appear with identical HTML around their DynamicLabels, the way the GridView control builds table cells around data fields labels for column titles.

  If you define any content in the **Template** property and intend to use the HeaderCell, add a PlaceHolder control where a list of HeaderCells will be inserted and set the **CellPlaceHolderID** property to the ID of that PlaceHolder control.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <HeaderRow Tag="value" CssClass="value" Style="value"
    NamedStyle="value" NamedStyleFallback="value" CellPlaceHolderID="ID"
    PatternTemplateName="name"  >
    <Template>
      HTML, DynamicLabels, DynamicColumnTitles, and other controls
      If using HeaderCells, include a PlaceHolder control where they will go
    </Template>
    <NamedParts>
      <desDD:HeaderRowNamedPatternPart Name="name" UseStandardContainer="boolean"
        Same properties as <HeaderRow> >
        <Template />
      </desDD:HeaderRowNamedPatternPart>
    </NamedParts>
  </HeaderRow>
</desDD:DynamicPartsPatterns>
```

Properties on this Part
Click on any of these topics to jump to them:

- Container Tag Properties
- Inner content Properties
- NamedParts Properties
- Behavior Properties

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

Example: DynamicColumnTitles within HeaderRow; HeaderCells not used

This supports 3 data fields.

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <HeaderRow Tag="Div" CssClass="HeaderRow" NamedStyle="HeaderRow" >
    <Template>
      <desDD:DynamicColumnTitle id="Title1" runat="server"/> | 
      <desDD:DynamicColumnTitle id="Title2" runat="server"/> | 
      <desDD:DynamicColumnTitle id="Title3" runat="server"/>
    </Template>
  </HeaderRow>
</desDD:DynamicPartsPatterns>
```

Example: HeaderCells for DynamicControls; HeaderRow provides the Table Row tag

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <HeaderRow Tag="Tr" CssClass="HeaderRow" NamedStyle="HeaderRow" >
  </HeaderRow>
  <HeaderCell Tag="Td" CssClass="HeaderCell" NamedStyle="HeaderCell" >
    <Template>
      <desDD:DynamicColumnTitle id="Title1" runat="server"/>
    </Template>
  </HeaderCell>
</desDD:DynamicPartsPatterns>
```

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

Example: HeaderCells for DynamicControls; HeaderRow provides a header

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <HeaderRow Tag="Div" CssClass="HeaderRow" NamedStyle="HeaderRow"
    CellPlaceHolderID="HeaderCellPlaceHolder" >
    <Template>
      <hr />  <!-- separator before the HeaderCells -->
      <asp:PlaceHolder id="HeaderCellPlaceHolder" runat="server" />
    </Template>
  </HeaderRow>
  <HeaderCell Tag="Div" CssClass="HeaderCell" NamedStyle="HeaderCell" >
    <Template>
      <desDD:DynamicColumnTitle id="Title1" runat="server"/>
    </Template>
  </HeaderCell>
</desDD:DynamicPartsPatterns>
```

Using this Part within a ListView control

On a ListView, use it in the **HeaderTemplate**.

```
<HeaderTemplate>
  <desDD:DynamicPattern id="DPHeaderRow" runat="server"
    PatternTemplateName="name" PartsToGenerate="HeaderRow"
    AutoGenerateButtons="First" />
</HeaderTemplate>
```

Using this Part within a FormView control

On a FormView, use it in the **HeaderTemplate**.

```
<HeaderTemplate>
  <desDD:DynamicPattern id="DPHeaderRow" runat="server"
    PatternTemplateName="name" PartsToGenerate="HeaderRow"
    AutoGenerateButtons="Last" />
</HeaderTemplate>
```

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

Using the HeaderCell Part

NamedPart class	HeaderCellNamedPatternPart
Use in ListView within	Generally used indirectly through the HeaderRow Part
Use in FormView within	Generally used indirectly through the HeaderRow Part
Available Templates	Template
Binds data fields to	DynamicColumnTitle, DynamicLabel
When using the DynamicPattern Control:	
PartsToGenerate values	HeaderCell, DetailsView
Other DynamicPattern properties	None

Describes the content of a single field's header, such as a column title. It usually has a DynamicColumnTitle or DynamicLabel which will automatically be assigned the data field name. When not setup, it generates a DynamicColumnTitle control as its content.

There are two ways to output data fields in DynamicColumnTitle and DynamicLabel controls.

- Define the complete HTML layout with DynamicColumnTitle within the HeaderRow. Do not use the HeaderCells.

  Use this when you have a layout that varies its appearance depending on the data field and you know exactly how many data fields will be offered. You will add exactly that number of DynamicColumnTitles.

- Define the inner content using the HeaderCell Part. Use the HeaderRow to establish a row header and footer than encloses all of the HeaderCell Parts. It determines the number of HeaderCells from the list of DataFieldInPattern objects within the **ItemsInPattern** collection on the DynamicPattern, DynamicListView, or DynamicFormView controls.

  Use this when all data fields will appear with identical HTML around their DynamicColumnTitle or DynamicLabel controls, the way the GridView control builds table cells around data field names for column titles.

  If you define any content in the **Template** property and intend to use the HeaderCell, add a PlaceHolder control where a list of HeaderCells will be inserted and set the **CellPlaceHolderID** property to the ID of that PlaceHolder control.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <HeaderCell Tag="value" CssClass="value" Style="value"
    NamedStyle="value" NamedStyleFallback="value"
    PatternTemplateName="name" >
    <Template>
      HTML, DynamicColumnTitles, DynamicLabels, and other controls
    </Template>
    <NamedParts>
      <desDD:HeaderCellNamedPatternPart Name="name" UseStandardContainer="boolean"
        Same properties as <HeaderCell> >
        <Template />
      </desDD:HeaderCellNamedPatternPart>
    </NamedParts>
  </HeaderRow>
</desDD:DynamicPartsPatterns>
```

Properties on this Part
Click on any of these topics to jump to them:

- Container Tag Properties
- Inner content Properties
- NamedParts Properties
- Behavior Properties

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

Example: HeaderCells for DynamicControls; HeaderRow provides the Table Row tag

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <HeaderRow Tag="Tr" CssClass="HeaderRow" NamedStyle="HeaderRow" >
  </HeaderRow>
  <HeaderCell Tag="Td" CssClass="HeaderCell" NamedStyle="HeaderCell" >
    <Template>
      <desDD:DynamicControl id="FieldData1" runat="server"/>
    </Template>
  </HeaderCell>
</desDD:DynamicPartsPatterns>
```

Example: HeaderCells for DynamicControls; HeaderRow provides a header

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <HeaderRow Tag="Div" CssClass="HeaderRow" NamedStyle="HeaderRow"
    CellPlaceHolderID="HeaderCellPlaceHolder" >
    <Template>
      <hr />  <!-- separator before the HeaderCells -->
      <asp:PlaceHolder id="HeaderCellPlaceHolder" runat="server" />
    </Template>
  </HeaderRow>
  <HeaderCell Tag="Div" CssClass="HeaderCell" NamedStyle="HeaderCell" >
    <Template>
      <desDD:DynamicControl id="FieldData1" runat="server"/>
    </Template>
  </HeaderCell>
</desDD:DynamicPartsPatterns>
```

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

Using this Part within a ListView control

Generally you will use the HeaderRow, not the HeaderCell, from within a ListView control. However, you can use HeaderCell so long as you supply the **DataField** property.

On a ListView, use it in the **HeaderTemplate** property.

```
<HeaderTemplate>
  <desDD:DynamicPattern id="DPHdrProductName" runat="server"
    PatternTemplateName="name" PartsToGenerate="HeaderCell"
    DataField="ProductName" /> | 
  <desDD:DynamicPattern id="DPHdrUnitPrice" runat="server"
    PatternTemplateName="name" PartsToGenerate="HeaderCell"
    DataField="UnitPrice" /> | 
  <desDD:DynamicPattern id="DPHdrDescription" runat="server"
    PatternTemplateName="name" PartsToGenerate="HeaderCell"
    DataField="Description" />
  <desDD:DynamicPattern id="DPHdrButtons" runat="server"
    PatternTemplateName="name" PartsToGenerate="HeaderButtons" />
</HeaderTemplate>
```

Using this Part within a FormView control

In the unlikely case where you need all column titles to displayed before the data fields in a FormView, use the same methodology as with ListView, above.

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

Using the HeaderButtons Part

NamedPart class	HeaderButtonsNamedPatternPart
Use in ListView within	Generally used indirectly through the HeaderRow Part
Use in FormView within	Generally used indirectly through the HeaderRow Part
Available Templates	Template
Binds data fields to	None
When using the DynamicPattern Control:	
PartsToGenerate values	HeaderButtons, DetailsView
Other DynamicPattern properties	None

Describes a header for a column containing Buttons parts. It is often the column title for buttons. When not setup, it uses " ".

This part is only used when the **HeaderRow** also allows the output of **HeaderCells**. It depends on either the **AutoGenerateButtons** property on DynamicPattern, DynamicListView, or DynamicFormView controls or on the presence of a ButtonsInPattern object in the **ItemsInPattern** collection. The ButtonsInPattern object specifies an alternative column title in its **HeaderText** property.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <HeaderButtons Tag="value" CssClass="value" Style="value"
    NamedStyle="value" NamedStyleFallback="value"
    PatternTemplateName="name" >
    <Template>
      HTML, DynamicLabel control, and other controls
    </Template>
    <NamedParts>
      <desDD:HeaderButtonsNamedPatternPart Name="name" UseStandardContainer="boolean"
        Same properties as <HeaderButtons> >
        <Template />
      </desDD:HeaderButtonsNamedPatternPart>
    </NamedParts>
  </HeaderButtons>
</desDD:DynamicPartsPatterns>
```

Click on any of these topics to jump to them:

- Container Tag Properties
- Inner content Properties
- NamedParts Properties
- Behavior Properties

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

Example

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
   <HeaderRow Tag="Tr" CssClass="HeaderRow" NamedStyle="HeaderRow" >
   </HeaderRow>
   <HeaderCell Tag="Td" CssClass="HeaderCell" NamedStyle="HeaderCell" >
      <Template>
         <desDD:DynamicColumnTitle id="Title1" runat="server"/>
      </Template>
   </HeaderCell>
   <HeaderButtons Tag="Td" CssClass="HeaderButtons" NamedStyle="HeaderButtons" >
      <Template>
         Commands
      </Template>
   </HeaderButtons>
</desDD:DynamicPartsPatterns>
```

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

Using this Part within a ListView control

Generally you will use the HeaderRow Part, not the HeaderButtons Part, from within a ListView control. The DynamicPattern control will use the **AutoGenerateButtons** property or a ButtonsInPattern object in the **ItemsInPattern** property to request the HeaderButtons Part. However, you can use the HeaderButtons part as shown here.

On a ListView, use it in the **HeaderTemplate** property.

```
<HeaderTemplate>
  <desDD:DynamicPattern id="DPHdrProductName" runat="server"
    PatternTemplateName="name" PartsToGenerate="HeaderCell"
    DataField="ProductName" /> | 
  <desDD:DynamicPattern id="DPHdrUnitPrice" runat="server"
    PatternTemplateName="name" PartsToGenerate="HeaderCell"
    DataField="UnitPrice" /> | 
  <desDD:DynamicPattern id="DPHdrDescription" runat="server"
    PatternTemplateName="name" PartsToGenerate="HeaderCell"
    DataField="Description" />
  <desDD:DynamicPattern id="DPHdrButtons" runat="server"
    PatternTemplateName="name" PartsToGenerate="HeaderButtons" />
</HeaderTemplate>
```

Using this Part within a FormView control

In the unlikely case where you need all column titles to displayed before the data fields in a FormView, use the same methodology as with ListView, above.

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

Using the FooterRow Part

NamedPart class	FooterRowNamedPatternPart
Use in ListView within	FooterTemplate
Use in FormView within	FooterTemplate
Available Templates	Template
Binds data fields to	DynamicLabel
When using the DynamicPattern Control:	
PartsToGenerate values	FooterRow, DetailsView
Other DynamicPattern properties	AutoGenerateButtons

Describes the content for a footer. The text for these labels comes from the **FooterText** property on the DataFieldInPattern or LabelInPattern objects.

There are two ways to output DynamicLabels.

- Define the complete HTML layout with DynamicLabels within the FooterRow. Do not use the FooterCells.

  Use this when you have a layout that varies its appearance depending on the data field and you know exactly how many data fields will be offered. You will add exactly that number of DynamicLabels.

- Define the inner content using the FooterCell Part. Use the FooterRow to establish a row header and footer than encloses all of the FooterCell Parts. It determines the number of FooterCells from the list of DataFieldInPattern objects within the **ItemsInPattern** collection on the DynamicPattern, DynamicListView, or DynamicFormView controls.

  Use this when all labels will appear with identical HTML around their DynamicLabels, the way the GridView control builds table cells around data fields labels for column titles.

  If you define any content in the **Template** property and intend to use the FooterCell, add a PlaceHolder control where a list of FooterCells will be inserted and set the **CellPlaceHolderID** property to the ID of that PlaceHolder control.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <FooterRow Tag="value" CssClass="value" Style="value"
    NamedStyle="value" NamedStyleFallback="value" CellPlaceHolderID="ID"
    PatternTemplateName="name"  >
    <Template>
      HTML, DynamicLabels, DynamicColumnTitles, and other controls
      If using FooterCells, include a PlaceHolder control where they will go
    </Template>
    <NamedParts>
      <desDD:FooterRowNamedPatternPart Name="name" UseStandardContainer="boolean"
        Same properties as <FooterRow> >
        <Template />
      </desDD:FooterRowNamedPatternPart>
    </NamedParts>
  </FooterRow>
</desDD:DynamicPartsPatterns>
```

Properties on this Part
Click on any of these topics to jump to them:

- Container Tag Properties
- Inner content Properties
- NamedParts Properties
- Behavior Properties

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

Example: DynamicColumnTitles within FooterRow; FooterCells not used

This supports 3 data fields.

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <FooterRow Tag="Div" CssClass="FooterRow" NamedStyle="FooterRow" >
    <Template>
      <desDD:DynamicLabel id="FooterLabel1" runat="server"/> | 
      <desDD:DynamicLabel id="FooterLabel2" runat="server"/> | 
      <desDD:DynamicLabel id="FooterLabel3" runat="server"/>
    </Template>
  </FooterRow>
</desDD:DynamicPartsPatterns>
```

Example: FooterCells for DynamicControls; FooterRow provides the Table Row tag

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <FooterRow Tag="Tr" CssClass="FooterRow" NamedStyle="FooterRow" >
  </FooterRow>
  <FooterCell Tag="Td" CssClass="FooterCell" NamedStyle="FooterCell" >
    <Template>
      <desDD:DynamicLabel id="FooterLabel1" runat="server"/>
    </Template>
  </FooterCell>
</desDD:DynamicPartsPatterns>
```

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

Example: FooterCells for DynamicControls; FooterRow provides a header

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <FooterRow Tag="Div" CssClass="FooterRow" NamedStyle="FooterRow"
    CellPlaceHolderID="FooterCellPlaceHolder" >
    <Template>
      <hr />  <!-- separator before the FooterCells -->
      <asp:PlaceHolder id="FooterCellPlaceHolder" runat="server" />
    </Template>
  </FooterRow>
  <FooterCell Tag="Div" CssClass="FooterCell" NamedStyle="FooterCell" >
    <Template>
      <desDD:DynamicLabel id="FooterLabel1" runat="server"/>
    </Template>
  </FooterCell>
</desDD:DynamicPartsPatterns>
```

Using this Part within a ListView control

On a ListView, use it in the **FooterTemplate**.

```
<FooterTemplate>
  <desDD:DynamicPattern id="DPFooterRow" runat="server"
    PatternTemplateName="name" PartsToGenerate="FooterRow"
    AutoGenerateButtons="First" />
</FooterTemplate>
```

Using this Part within a FormView control

On a FormView, use it in the **FooterTemplate**.

```
<FooterTemplate>
  <desDD:DynamicPattern id="DPFooterRow" runat="server"
    PatternTemplateName="name" PartsToGenerate="FooterRow"
    AutoGenerateButtons="Last" />
</FooterTemplate>
```

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

Using the FooterCell Part

NamedPart class	FooterCellNamedPatternPart
Use in ListView within	Generally used indirectly through the FooterRow Part
Use in FormView within	Generally used indirectly through the FooterRow Part
Available Templates	Template
Binds data fields to	DynamicLabel
When using the DynamicPattern Control:	
PartsToGenerate values	FooterCell, DetailsView
Other DynamicPattern properties	None

Describes the content of a single field's footer. It usually has a DynamicLabel which will automatically be assigned the data field name. When not setup, it generates a DynamicLabel control as its content.

There are two ways to output DynamicLabel controls.

- Define the complete HTML layout with DynamicLabel within the FooterRow. Do not use the FooterCells.

  Use this when you have a layout that varies its appearance depending on the data field and you know exactly how many data fields will be offered. You will add exactly that number of DynamicLabels.

- Define the inner content using the FooterCell Part. Use the FooterRow to establish a row header and footer than encloses all of the FooterCell Parts. It determines the number of FooterCells from the list of DataFieldInPattern objects within the **ItemsInPattern** collection on the DynamicPattern, DynamicListView, or DynamicFormView controls.

  Use this when all data fields will appear with identical HTML around their DynamicLabel controls, the way the GridView control builds table cells around data field names for column titles.

  If you define any content in the **Template** property and intend to use the FooterCell, add a PlaceHolder control where a list of FooterCells will be inserted and set the **CellPlaceHolderID** property to the ID of that PlaceHolder control.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <FooterCell Tag="value" CssClass="value" Style="value"
    NamedStyle="value" NamedStyleFallback="value"
    PatternTemplateName="name" >
    <Template>
      HTML, DynamicLabels, and other controls
    </Template>
    <NamedParts>
      <desDD:FooterCellNamedPatternPart Name="name" UseStandardContainer="boolean"
        Same properties as <FooterCell> >
        <Template />
      </desDD:FooterCellNamedPatternPart>
    </NamedParts>
  </FooterRow>
</desDD:DynamicPartsPatterns>
```

Properties on this Part
Click on any of these topics to jump to them:

- Container Tag Properties
- Inner content Properties
- NamedParts Properties
- Behavior Properties

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

Example: FooterCells for DynamicControls; FooterRow provides the Table Row tag

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <FooterRow Tag="Tr" CssClass="FooterRow" NamedStyle="FooterRow" >
  </FooterRow>
  <FooterCell Tag="Td" CssClass="FooterCell" NamedStyle="FooterCell" >
    <Template>
      <desDD:DynamicLabel id="FooterLabel1" runat="server"/>
    </Template>
  </FooterCell>
</desDD:DynamicPartsPatterns>
```

Example: FooterCells for DynamicControls; FooterRow provides a header

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <FooterRow Tag="Div" CssClass="FooterRow" NamedStyle="FooterRow"
    CellPlaceHolderID="FooterCellPlaceHolder" >
    <Template>
      <hr />  <!-- separator before the FooterCells -->
      <asp:PlaceHolder id="FooterCellPlaceHolder" runat="server" />
    </Template>
  </FooterRow>
  <FooterCell Tag="Div" CssClass="FooterCell" NamedStyle="FooterCell" >
    <Template>
      <desDD:DynamicLabel id="FooterLabel1" runat="server"/>
    </Template>
  </FooterCell>
</desDD:DynamicPartsPatterns>
```

Using this Part within a ListView control

Generally you will use the FooterRow, not the FooterCell, from within a ListView control. However, you can use FooterCell so long as you supply the **DataField** property.

On a ListView, use it in the **FooterTemplate** property.

```
<FooterTemplate>
  <desDD:DynamicPattern id="DPFtrProductName" runat="server"
    PatternTemplateName="name" PartsToGenerate="FooterCell" >
    <ItemsInPattern>
      <desDD:LabelInPattern FooterText="<hr />" />
    </ItemsInPattern>
  </desDD:DynamicPattern>
  <desDD:DynamicPattern id="DPFtrUnitPrice" runat="server"
    PatternTemplateName="name" PartsToGenerate="FooterCell" >
    <ItemsInPattern>
      <desDD:LabelInPattern FooterText="<hr />" />
    </ItemsInPattern>
  </desDD:DynamicPattern>
  <desDD:DynamicPattern id="DPFtrDescription" runat="server"
    PatternTemplateName="name" PartsToGenerate="FooterCell" >
    <ItemsInPattern>
      <desDD:LabelInPattern FooterText="<hr />" />
    </ItemsInPattern>
  </desDD:DynamicPattern>
  <desDD:DynamicPattern id="DPFtrButtons" runat="server"
    PatternTemplateName="name" PartsToGenerate="FooterButtons" />
</FooterTemplate>
```

Using this Part within a FormView control

In the unlikely case where you need all column titles to displayed before the data fields in a FormView, use the same methodology as with ListView, above.

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

Using the FooterButtons Part

NamedPart class	FooterButtonsNamedPatternPart
Use in ListView within	Generally used indirectly through the FooterRow Part
Use in FormView within	Generally used indirectly through the FooterRow Part
Available Templates	Template
Binds data fields to	None
When using the DynamicPattern Control:	
PartsToGenerate values	FooterButtons, DetailsView
Other DynamicPattern properties	None

Describes a header for a column containing Buttons parts. When not setup, it uses "` `".

This part is only used when the **FooterRow** also allows the output of **FooterCells**. It depends on either the **AutoGenerateButtons** property on DynamicPattern, DynamicListView, or DynamicFormView controls or on the presence of a ButtonsInPattern object in the **ItemsInPattern** collection. The ButtonsInPattern object specifies an alternative column title in its **FooterText** property.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <FooterButtons Tag="value" CssClass="value" Style="value"
    NamedStyle="value" NamedStyleFallback="value"
    PatternTemplateName="name" >
    <Template>
      HTML, DynamicLabel control, and other controls
    </Template>
    <NamedParts>
      <desDD:FooterButtonsNamedPatternPart Name="name" UseStandardContainer="boolean"
        Same properties as <FooterButtons> >
        <Template />
      </desDD:FooterButtonsNamedPatternPart>
    </NamedParts>
  </FooterButtons>
</desDD:DynamicPartsPatterns>
```

Click on any of these topics to jump to them:

- Container Tag Properties
- Inner content Properties
- NamedParts Properties
- Behavior Properties

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

Example

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <FooterRow Tag="Tr" CssClass="FooterRow" NamedStyle="FooterRow" >
  </FooterRow>
  <FooterCell Tag="Td" CssClass="FooterCell" NamedStyle="FooterCell" >
    <Template>
      <desDD:DynamicLabel id="FooterText1" runat="server"/>
    </Template>
  </FooterCell>
  <FooterButtons Tag="Td" CssClass="FooterButtons" NamedStyle="FooterButtons" >
    <Template>
      Commands
    </Template>
  </FooterButtons>
</desDD:DynamicPartsPatterns>
```

Using this Part within a ListView control

Generally you will use the FooterRow Part, not the FooterButtons Part, from within a ListView control. The DynamicPattern control will use the **AutoGenerateButtons** property or a ButtonsInPattern object in the **ItemsInPattern** property to request the FooterButtons Part. However, you can use the FooterButtons part as shown here.

On a ListView, use it in the **FooterTemplate** property.

```
<FooterTemplate>
  <desDD:DynamicPattern id="DPFtrProductName" runat="server"
    PatternTemplateName="name" PartsToGenerate="FooterCell" >
    <ItemsInPattern>
      <desDD:LabelInPattern FooterText="<hr />" />
    </ItemsInPattern>
  </desDD:DynamicPattern>
  <desDD:DynamicPattern id="DPFtrUnitPrice" runat="server"
    PatternTemplateName="name" PartsToGenerate="FooterCell" >
    <ItemsInPattern>
      <desDD:LabelInPattern FooterText="<hr />" />
    </ItemsInPattern>
  </desDD:DynamicPattern>
  <desDD:DynamicPattern id="DPFtrDescription" runat="server"
    PatternTemplateName="name" PartsToGenerate="FooterCell" >
    <ItemsInPattern>
      <desDD:LabelInPattern FooterText="<hr />" />
    </ItemsInPattern>
  </desDD:DynamicPattern>
  <desDD:DynamicPattern id="DPFtrButtons" runat="server"
    PatternTemplateName="name" PartsToGenerate="FooterButtons" />
</FooterTemplate>
```

Using this Part within a FormView control

In the unlikely case where you need all column titles to displayed before the data fields in a FormView, use the same methodology as with ListView, above.

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

Using the NoDataToShow Part

NamedPart class	NoDataToShowNamedPatternPart
Use in ListView within	EmptyDataTemplate
Use in FormView within	EmptyDataTemplate
Available Templates	Template
Binds data fields to	None
When using the DynamicPattern Control:	
PartsToGenerate values	NoDataToShow
Other DynamicPattern properties	None

Used when there is no data found. It is often a message like "No data found." The message can either be defined within the **Template** property or as a string in the **Message** property.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <NoDataToShow Tag="value" CssClass="value" Style="value"
    NamedStyle="value" NamedStyleFallback="value"
    PatternTemplateName="name" Message="message" MessageLookupID="value" >
    <Template>
      HTML and other controls
    </Template>
    <NamedParts>
      <desDD:NoDataToShowNamedPatternPart Name="name" UseStandardContainer="boolean"
        Same properties as <NoDataToShow> >
        <Template />
      </desDD:NoDataToShowNamedPatternPart>
    </NamedParts>
  </NoDataToShow>
</desDD:DynamicPartsPatterns>
```

Click on any of these topics to jump to them:

- Container Tag Properties
- Inner content Properties
- NamedParts Properties
- Behavior Properties

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

Example: Using Message property

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <NoDataToShow Tag="Div" CssClass="NoDataToShow" NamedStyle="NoDataToShow"
    Message="No records found.">
    <Template>
      <desDD:DynamicLabel id="FooterLabel1" runat="server"/>
    </Template>
  </NoDataToShow>
</desDD:DynamicPartsPatterns>
```

Example: Using Template property

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <NoDataToShow Tag="Div" CssClass="NoDataToShow" NamedStyle="NoDataToShow" >
    <Template>
      No records found.<br />
      <des:Button id="ContinueBtn" runat="server" Text="Continue" />
    </Template>
  </NoDataToShow>
</desDD:DynamicPartsPatterns>
```

Using this Part within a ListView control

On a ListView, use it in the **EmptyDataTemplate**.

```
<EmptyDataTemplate>
  <desDD:DynamicPattern id="DPNoDataToShow" runat="server"
    PatternTemplateName="name" PartsToGenerate="NoDataToShow" />
</EmptyDataTemplate>
```

Using this Part within a FormView control

On a FormView, use it in the **EmptyDataTemplate**.

```
<EmptyDataTemplate>
  <desDD:DynamicPattern id="DPNoDataToShow" runat="server"
    PatternTemplateName="name" PartsToGenerate="NoDataToShow" />
</EmptyDataTemplate>
```

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

Using the DataRowSeparator Part

NamedPart class	DataRowSeparatorNamedPatternPart
Use in ListView within	ItemSeparatorTemplate
Use in FormView within	n/a
Available Templates	Template
Binds data fields to	None
When using the DynamicPattern Control:	
PartsToGenerate values	DataRowSeparator
Other DynamicPattern properties	None

A row that separates each record.

There are two ways to output content.

- Define the complete HTML layout within the DataRowSeparator. Do not use the DataRowSeparatorCells.
- Define the inner content using the DataRowSeparatorCell Part. Use the DataRowSeparator to establish a row header and footer than encloses all of the DataRowSeparatorCell Parts. It determines the number of DataRowSeparatorCells from the list of DataFieldInPattern objects within the **ItemsInPattern** collection on the DynamicPattern, DynamicListView, or DynamicFormView controls.

  If you define any content in the **Template** property and intend to use the DataRowSeparatorCell, add a PlaceHolder control where a list of DataRowSeparatorCells will be inserted and set the **CellPlaceHolderID** property to the ID of that PlaceHolder control.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <DataRowSeparator Tag="value" CssClass="value" Style="value"
    NamedStyle="value" NamedStyleFallback="value" CellPlaceHolderID="ID"
    PatternTemplateName="name"  >
    <Template>
      HTML and other controls
      If using DataRowSeparatorCells, include a PlaceHolder control
     where they will go
    </Template>
    <NamedParts>
      <desDD:DataRowSeparatorNamedPatternPart Name="name"
        UseStandardContainer="boolean"
        Same properties as <DataRowSeparator> >
        <Template />
      </desDD:DataRowSeparatorNamedPatternPart>
    </NamedParts>
  </DataRowSeparator>
</desDD:DynamicPartsPatterns>
```

Properties on this Part
Click on any of these topics to jump to them:

- Container Tag Properties
- Inner content Properties
- NamedParts Properties
- Behavior Properties

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

Example: DataRowSeparatorCells not used

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <DataRowSeparator Tag="Div" CssClass="DataRowSeparator"
    NamedStyle="DataRowSeparator" >
    <Template>
      <hr />
    </Template>
  </DataRowSeparator>
</desDD:DynamicPartsPatterns>
```

Example: DataRowSeparatorCells are used

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <DataRowSeparator Tag="Tr" CssClass="DataRowSeparator"
    NamedStyle="DataRowSeparator" >
  </DataRowSeparator>
  <DataRowSeparatorCell Tag="Td" CssClass="DataRowSeparatorCell"
    NamedStyle="DataRowSeparatorCell" >
    <Template>
      <hr />
    </Template>
  </DataRowSeparatorCell>
</desDD:DynamicPartsPatterns>
```

Example: Both DataRowSeparator and DataRowSeparatorCells provide content

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <DataRowSeparator Tag="Div" CssClass="DataRowSeparator"
    NamedStyle="DataRowSeparator"
    CellPlaceHolderID="DataRowSeparatorCellPlaceHolder" >
    <Template>
      <br />
      <asp:PlaceHolder id="DataRowSeparatorCellPlaceHolder" runat="server" />
      <br />
    </Template>
  </DataRowSeparator>
  <DataRowSeparatorCell Tag="Div" CssClass="DataRowSeparatorCell"
    NamedStyle="DataRowSeparatorCell" >
    <Template>
      <hr />
    </Template>
  </DataRowSeparatorCell>
</desDD:DynamicPartsPatterns>
```

Using this Part within a ListView control

On a ListView, use it in the **ItemSeparatorTemplate**.

```
<ItemSeparatorTemplate>
  <desDD:DynamicPattern id="DPDataRowSeparator" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataRowSeparator" />
</ItemSeparatorTemplate>
```

Using this Part within a FormView control

Not recommended. There is no Template associated with this Part.

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

Using the DataRowSeparatorCell Part

NamedPart class	DataRowSeparatorCellNamedPatternPart
Use in ListView within	Generally used indirectly through the DataRowSeparatorRow Part
Use in FormView within	n/a
Available Templates	Template
Binds data fields to	None
When using the DynamicPattern Control:	
PartsToGenerate values	DataRowSeparatorCell
Other DynamicPattern properties	None

A single field's content for the **DataRowSeparator**. When not setup, it generates "` `" as its content.

There are two ways to output content.

- Define the complete HTML layout within the DataRowSeparatorRow. Do not use the DataRowSeparatorCells.
- Define the inner content using the DataRowSeparatorCell Part. Use the DataRowSeparatorRow to establish a row header and footer than encloses all of the DataRowSeparatorCell Parts. It determines the number of DataRowSeparatorCells from the list of DataFieldInPattern objects within the **ItemsInPattern** collection on the DynamicPattern, DynamicListView, or DynamicFormView controls.

  If you define any content in the **Template** property and intend to use the DataRowSeparatorCell, add a PlaceHolder control where a list of DataRowSeparatorCells will be inserted and set the **CellPlaceHolderID** property to the ID of that PlaceHolder control.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <DataRowSeparatorCell Tag="value" CssClass="value" Style="value"
    NamedStyle="value" NamedStyleFallback="value"
    PatternTemplateName="name" >
    <Template>
      HTML, DynamicLabels, and other controls
    </Template>
    <NamedParts>
      <desDD:DataRowSeparatorCellNamedPatternPart Name="name"
        UseStandardContainer="boolean"
        Same properties as <DataRowSeparatorCell> >
        <Template />
      </desDD:DataRowSeparatorCellNamedPatternPart>
    </NamedParts>
  </DataRowSeparatorCell>
</desDD:DynamicPartsPatterns>
```

Properties on this Part
Click on any of these topics to jump to them:

- Container Tag Properties
- Inner content Properties
- NamedParts Properties
- Behavior Properties

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

Example: DataRowSeparatorCells not used

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <DataRowSeparator Tag="Div" CssClass="DataRowSeparator"
    NamedStyle="DataRowSeparator" >
    <Template>
      <hr />
    </Template>
  </DataRowSeparator>
</desDD:DynamicPartsPatterns>
```

Example: DataRowSeparatorCells are used

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <DataRowSeparator Tag="Tr" CssClass="DataRowSeparator"
    NamedStyle="DataRowSeparator" >
  </DataRowSeparator>
  <DataRowSeparatorCell Tag="Td" CssClass="DataRowSeparatorCell"
    NamedStyle="DataRowSeparatorCell" >
    <Template>
      <hr />
    </Template>
  </DataRowSeparatorCell>
</desDD:DynamicPartsPatterns>
```

Example: Both DataRowSeparator and DataRowSeparatorCells provide content

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <DataRowSeparator Tag="Div" CssClass="DataRowSep" NamedStyle="DataRowSep"
    CellPlaceHolderID="DataRowSepCellPlaceHolder" >
    <Template>
      <br />
      <asp:PlaceHolder id="DataRowSepCellPlaceHolder" runat="server" />
      <br />
    </Template>
  </DataRowSeparator>
  <DataRowSeparatorCell Tag="Div" CssClass="DataRowSepCell" NamedStyle="DataRowSepCell" >
    <Template>
      <hr />
    </Template>
  </DataRowSeparatorCell>
</desDD:DynamicPartsPatterns>
```

Using this Part within a ListView control

Generally you will use the DataRowSeparator Part, not the DataRowSeparatorCell Part, from within a ListView control. However, you can use the DataRowSeparatorCell part as shown here.

On a ListView, use it in the **FooterTemplate** property.

```
<FooterTemplate>
  <desDD:DynamicPattern id="DPSepProductName" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataRowSeparatorCell" />
  <desDD:DynamicPattern id="DPSepUnitPrice" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataRowSeparatorCell" />
  <desDD:DynamicPattern id="DPSepDescription" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataRowSeparatorCell" />
</FooterTemplate>
```

Using this Part within a FormView control

Not recommended. There is no Template associated with this Part.

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

Using the GroupContainer Part

NamedPart class	GroupContainerNamedPatternPart
Use in ListView within	LayoutTemplate
Use in FormView within	n/a
Available Templates	None
Binds data fields to	None
When using the DynamicPattern Control:	
PartsToGenerate values	GroupContainerOpeningTag, GroupContainerClosingTag
Other DynamicPattern properties	None

Associated with ListView's Grouping feature. When using DynamicListView, set **EnableGrouping** to `true`.

The HTML tag that encloses a group. Specify its **Tag** property to define the HTML tag. It does not supply a **Template** property for any of its inner content because the rest of the tags are the inner content.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <GroupContainer Tag="value" CssClass="value" Style="value"
    NamedStyle="value" NamedStyleFallback="value" >
    <NamedParts>
      <desDD:GroupContainerNamedPatternPart Name="name"
        UseStandardContainer="boolean"
        Same properties as <GroupContainer> >
      </desDD:GroupContainerNamedPatternPart>
    </NamedParts>
  </GroupContainer>
</desDD:DynamicPartsPatterns>
```

Properties on this Part
Click on any of these topics to jump to them:

- Container Tag Properties
- NamedParts Properties
- Behavior Properties

Using this Part within a ListView control

On a ListView, use it in the **LayoutTemplate**. Define two, one for the opening containing tag before everything else and another for the closing containing tag after. When using the ListView's grouping feature, setup the **GroupTemplate** to reflect the content that was previously in the **LayoutTemplate**.

```
<LayoutTemplate>
  <desDD:DynamicPattern id="DPGrpOpening" runat="server"
    PatternTemplateName="name" PartsToGenerate="GroupContainerOpeningTag" />
  HTML and controls to generate the headers, data rows and footers
  <desDD:DynamicPattern id="DPGrpClosing" runat="server"
    PatternTemplateName="name" PartsToGenerate="GroupContainerClosingTag" />
</LayoutTemplate>
<GroupTemplate>
  <desDD:DynamicPattern id="DPOpening" runat="server"
    PatternTemplateName="name" PartsToGenerate="ContainerOpeningTag" />
  HTML and controls to generate the headers, data rows and footers
  <desDD:DynamicPattern id="DPClosing" runat="server"
    PatternTemplateName="name" PartsToGenerate="ContainerClosingTag" />
</GroupTemplate>
```

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

Using this Part within a FormView control

Not recommended. There is no Template associated with this Part.

Using this Part within a DynamicListView control

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
   DataSourceID="value" PatternTemplateName="value" EnableGrouping="true"
   GroupItemCount="number of tiled columns" ShowDataRowSeparator="true" />
```

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

Using the GroupHeader Part

NamedPart class	GroupHeaderNamedPatternPart
Use in ListView within	LayoutTemplate
Use in FormView within	n/a
Available Templates	Template
Binds data fields to	None
When using the DynamicPattern Control:	
PartsToGenerate values	GroupHeader
Other DynamicPattern properties	None

The header for the grouping feature. HTML that separates the **GroupContainer's** opening tag from the first Item of actual data. It is added in the **LayoutTemplate** property between the GroupContainerOpeningTag and the ItemPlaceHolder.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <GroupHeader Tag="value" CssClass="value" Style="value"
    NamedStyle="value" NamedStyleFallback="value"
    PatternTemplateName="name">
    <Template>
      HTML and other controls
    </Template>
    <NamedParts>
      <desDD:GroupHeaderNamedPatternPart Name="name" UseStandardContainer="boolean"
        Same properties as <GroupHeader> >
        <Template />
      </desDD:GroupHeaderNamedPatternPart>
    </NamedParts>
  </GroupHeader>
</desDD:DynamicPartsPatterns>
```

Click on any of these topics to jump to them:

- Container Tag Properties
- Inner content Properties
- NamedParts Properties
- Behavior Properties

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

Example

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <GroupHeader Tag="None">
    <Template>
      <tr><td>
    </Template>
  </GroupHeader>
  <GroupFooter Tag="None" >
    <Template>
      </td></tr>
    </Template>
  </GroupFooter>
  <GroupSeparator Tag="None" >
    <Template>
      </td></tr><tr><td>
    </Template>
  </GroupSeparator>
  <DataRowSeparator>
    <Template>
      </div></td><td><div<%= GetTagAttributes("Container", "Container") %>>
    </Template>
  </GroupSeparator>
</desDD:DynamicPartsPatterns>
```

Using this Part within a ListView control

On a ListView, use it in the **LayoutTemplate**. Also setup the **GroupSeparatorTemplate** for the GroupSeparator Part and the **ItemSeparatorTemplate** for the DataRowSeparator. Set the **GroupItemCount** to the number of items to show in tiled fashion.

```
<LayoutTemplate>
  <desDD:DynamicPattern id="DPGrpOpening" runat="server"
    PatternTemplateName="name" PartsToGenerate="GroupContainerOpeningTag" />
  <desDD:DynamicPattern id="DPGrpHeader" runat="server"
    PatternTemplateName="name" PartsToGenerate="GroupHeader" />
  HTML and controls to generate the data rows
  <desDD:DynamicPattern id="DPGrpFooter" runat="server"
    PatternTemplateName="name" PartsToGenerate="GroupFooter" />
  <desDD:DynamicPattern id="DPGrpClosing" runat="server"
    PatternTemplateName="name" PartsToGenerate="GroupContainerClosingTag" />
</LayoutTemplate>
<GroupSeparatorTemplate>
  <desDD:DynamicPattern id="DPGroupSeparator" runat="server"
    PatternTemplateName="name" PartsToGenerate="GroupSeparator" />
</GroupSeparatorTemplate>
<ItemSeparatorTemplate>
  <desDD:DynamicPattern id="DPItemSeparator" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataRowSeparator" />
</ItemSeparatorTemplate>
```

Using this Part within a FormView control

Not recommended. There is no Template associated with this Part.

Using this Part within a DynamicListView control

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
  DataSourceID="value" PatternTemplateName="value" EnableGrouping="true"
  GroupItemCount="number of tiled columns" ShowDataRowSeparator="true" />
```

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

Using the GroupFooter Part

NamedPart class	GroupFooterNamedPatternPart
Use in ListView within	LayoutTemplate
Use in FormView within	n/a
Available Templates	Template
Binds data fields to	None
When using the DynamicPattern Control:	
PartsToGenerate values	GroupFooter
Other DynamicPattern properties	None

The Footer for the grouping feature. HTML that separates the last Item of actual data from the **GroupContainer**'s closing tag. It is added in the **LayoutTemplate** property between the ItemPlaceHolder and the GroupContainerClosingTag.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <GroupFooter Tag="value" CssClass="value" Style="value"
    NamedStyle="value" NamedStyleFallback="value"
    PatternTemplateName="name">
    <Template>
      HTML and other controls
    </Template>
    <NamedParts>
      <desDD:GroupFooterNamedPatternPart Name="name" UseStandardContainer="boolean"
        Same properties as <GroupFooter> >
        <Template />
      </desDD:GroupFooterNamedPatternPart>
    </NamedParts>
  </GroupFooter>
</desDD:DynamicPartsPatterns>
```

Click on any of these topics to jump to them:

- Container Tag Properties
- Inner content Properties
- NamedParts Properties
- Behavior Properties

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

Example

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <GroupHeader Tag="None">
    <Template>
      <tr><td>
    </Template>
  </GroupHeader>
  <GroupFooter Tag="None" >
    <Template>
      </td></tr>
    </Template>
  </GroupFooter>
  <GroupSeparator Tag="None" >
    <Template>
      </td></tr><tr><td>
    </Template>
  </GroupSeparator>
  <DataRowSeparator>
    <Template>
      </div></td><td><div<%= GetTagAttributes("Container", "Container") %>>
    </Template>
  </GroupSeparator>
</desDD:DynamicPartsPatterns>
```

Using this Part within a ListView control

On a ListView, use it in the **LayoutTemplate**. Also setup the **GroupSeparatorTemplate** for the GroupSeparator Part and the **ItemSeparatorTemplate** for the DataRowSeparator. Set the **GroupItemCount** to the number of items to show in tiled fashion.

```
<LayoutTemplate>
  <desDD:DynamicPattern id="DPGrpOpening" runat="server"
    PatternTemplateName="name" PartsToGenerate="GroupContainerOpeningTag" />
  <desDD:DynamicPattern id="DPGrpHeader" runat="server"
    PatternTemplateName="name" PartsToGenerate="GroupHeader" />
  HTML and controls to generate the data rows
  <desDD:DynamicPattern id="DPGrpFooter" runat="server"
    PatternTemplateName="name" PartsToGenerate="GroupFooter" />
  <desDD:DynamicPattern id="DPGrpClosing" runat="server"
    PatternTemplateName="name" PartsToGenerate="GroupContainerClosingTag" />
</LayoutTemplate>
<GroupSeparatorTemplate>
  <desDD:DynamicPattern id="DPGroupSeparator" runat="server"
    PatternTemplateName="name" PartsToGenerate="GroupSeparator" />
</GroupSeparatorTemplate>
<ItemSeparatorTemplate>
  <desDD:DynamicPattern id="DPItemSeparator" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataRowSeparator" />
</ItemSeparatorTemplate>
```

Using this Part within a FormView control

Not recommended. There is no Template associated with this Part.

Using this Part within a DynamicListView control

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
  DataSourceID="value" PatternTemplateName="value" EnableGrouping="true"
  GroupItemCount="number of tiled columns" ShowDataRowSeparator="true" />
```

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

Using the GroupSeparator Part

NamedPart class	GroupSeparatorNamedPatternPart
Use in ListView within	GroupSeparatorTemplate
Use in FormView within	n/a
Available Templates	Template
Binds data fields to	None
When using the DynamicPattern Control:	
PartsToGenerate values	GroupSeparator
Other DynamicPattern properties	None

HTML that separates groups from each other when using ListView and DynamicListView. Associated with the **ListView.GroupSeparatorTemplate** property.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <GroupSeparator Tag="value" CssClass="value" Style="value"
    NamedStyle="value" NamedStyleFallback="value"
    PatternTemplateName="name">
    <Template>
      HTML and other controls
    </Template>
    <NamedParts>
      <desDD:GroupSeparatorNamedPatternPart Name="name"
        UseStandardContainer="boolean"
        Same properties as <GroupSeparator> >
        <Template />
      </desDD:GroupSeparatorNamedPatternPart>
    </NamedParts>
  </GroupSeparator>
</desDD:DynamicPartsPatterns>
```

Click on any of these topics to jump to them:

- Container Tag Properties
- Inner content Properties
- NamedParts Properties
- Behavior Properties

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

Example

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <GroupHeader Tag="None">
    <Template>
      <tr><td>
    </Template>
  </GroupHeader>
  <GroupSeparator Tag="None" >
    <Template>
      </td></tr>
    </Template>
  </GroupSeparator>
  <GroupSeparator Tag="None" >
    <Template>
      </td></tr><tr><td>
    </Template>
  </GroupSeparator>
  <DataRowSeparator>
    <Template>
      </div></td><td><div<%= GetTagAttributes("Container", "Container") %>>
    </Template>
  </GroupSeparator>
</desDD:DynamicPartsPatterns>
```

Using this Part within a ListView control

On a ListView, use it in the **LayoutTemplate**. Also setup the **GroupSeparatorTemplate** for the GroupSeparator Part and the **ItemSeparatorTemplate** for the DataRowSeparator. Set the **GroupItemCount** to the number of items to show in tiled fashion.

```
<LayoutTemplate>
  <desDD:DynamicPattern id="DPGrpOpening" runat="server"
    PatternTemplateName="name" PartsToGenerate="GroupContainerOpeningTag" />
  <desDD:DynamicPattern id="DPGrpHeader" runat="server"
    PatternTemplateName="name" PartsToGenerate="GroupHeader" />
  HTML and controls to generate the data rows
  <desDD:DynamicPattern id="DPGrpFooter" runat="server"
    PatternTemplateName="name" PartsToGenerate="GroupSeparator" />
  <desDD:DynamicPattern id="DPGrpClosing" runat="server"
    PatternTemplateName="name" PartsToGenerate="GroupContainerClosingTag" />
</LayoutTemplate>
<GroupSeparatorTemplate>
  <desDD:DynamicPattern id="DPGroupSeparator" runat="server"
    PatternTemplateName="name" PartsToGenerate="GroupSeparator" />
</GroupSeparatorTemplate>
<ItemSeparatorTemplate>
  <desDD:DynamicPattern id="DPItemSeparator" runat="server"
    PatternTemplateName="name" PartsToGenerate="DataRowSeparator" />
</ItemSeparatorTemplate>
```

Using this Part within a FormView control

Not recommended. There is no Template associated with this Part.

Using this Part within a DynamicListView control

```
<desDD:DynamicListView ID="DynamicListView1" runat="server"
  DataSourceID="value" PatternTemplateName="value" EnableGrouping="true"
  GroupItemCount="number of tiled columns" ShowDataRowSeparator="true" />
```

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

Using the EmptyItem Part

NamedPart class	EmptyItemNamedPatternPart
Use in ListView within	EmptyItemTemplate
Use in FormView within	n/a
Available Templates	Template
Binds data fields to	None
When using the DynamicPattern Control:	
PartsToGenerate values	EmptyItem
Other DynamicPattern properties	None

Defines the appearance where an item would normally go in a group, but there is no data for that item. Associated with **ListView.EditItemTemplate**.

There are two ways to output content.

- Define the complete HTML layout within the EmptyItem. Do not use the EmptyItemCells.
- Define the inner content using the EmptyItemCell Part. Use the EmptyItem to establish a row header and footer than encloses all of the EmptyItemCell Parts. It determines the number of EmptyItemCells from the list of DataFieldInPattern objects within the **ItemsInPattern** collection on the DynamicPattern, DynamicListView, or DynamicFormView controls.

  If you define any content in the **Template** property and intend to use the EmptyItemCell, add a PlaceHolder control where a list of EmptyItemCells will be inserted and set the **CellPlaceHolderID** property to the ID of that PlaceHolder control.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <EmptyItem Tag="value" CssClass="value" Style="value"
    NamedStyle="value" NamedStyleFallback="value" CellPlaceHolderID="ID"
    PatternTemplateName="name"  >
    <Template>
      HTML and other controls
      If using EmptyItemCells, include a PlaceHolder control
     where they will go
    </Template>
    <NamedParts>
      <desDD:EmptyItemNamedPatternPart Name="name"
        UseStandardContainer="boolean"
        Same properties as <EmptyItem> >
        <Template />
      </desDD:EmptyItemNamedPatternPart>
    </NamedParts>
  </EmptyItem>
</desDD:DynamicPartsPatterns>
```

Properties on this Part
Click on any of these topics to jump to them:

- Container Tag Properties
- Inner content Properties
- NamedParts Properties
- Behavior Properties

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

Example: EmptyItemCells not used

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <EmptyItem Tag="Div" CssClass="EmptyItem" NamedStyle="EmptyItem" >
    <Template>
       
    </Template>
  </EmptyItem>
</desDD:DynamicPartsPatterns>
```

Example: EmptyItemCells are used

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <EmptyItem Tag="Tr" CssClass="EmptyItem" NamedStyle="EmptyItem" >
  </EmptyItem>
  <EmptyItemCell Tag="Td" CssClass="EmptyItemCell" NamedStyle="EmptyItemCell" >
    <Template>
       
    </Template>
  </EmptyItemCell>
</desDD:DynamicPartsPatterns>
```

Example: Both EmptyItem and EmptyItemCells provide content

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <EmptyItem Tag="Div" CssClass="EmptyItem" NamedStyle="EmptyItem"
    CellPlaceHolderID="EmptyItemCellPlaceHolder" >
    <Template>
      <br />
      <asp:PlaceHolder id="EmptyItemCellPlaceHolder" runat="server" />
      <br />
    </Template>
  </EmptyItem>
  <EmptyItemCell Tag="Div" CssClass="EmptyItemCell" NamedStyle="EmptyItemCell" >
    <Template>
      <hr />
    </Template>
  </EmptyItemCell>
</desDD:DynamicPartsPatterns>
```

Using this Part within a ListView control

On a ListView, use it in the **EmptyItemTemplate**.

```
<EmptyItemTemplate>
  <desDD:DynamicPattern id="DPEmptyItem" runat="server"
    PatternTemplateName="name" PartsToGenerate="EmptyItem" />
</EmptyItemTemplate>
```

Using this Part within a FormView control

Not recommended. There is no Template associated with this Part.

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

Using the EmptyItemCell Part

NamedPart class	EmptyItemCellNamedPatternPart
Use in ListView within	Generally used indirectly through the EmptyItemRow Part
Use in FormView within	n/a
Available Templates	Template
Binds data fields to	None
When using the DynamicPattern Control:	
PartsToGenerate values	EmptyItemCell
Other DynamicPattern properties	None

A single field's content for the **EmptyItem**. When not setup, it generates "` `" as its content.

There are two ways to output content.

- Define the complete HTML layout within the EmptyItemRow. Do not use the EmptyItemCells.
- Define the inner content using the EmptyItemCell Part. Use the EmptyItemRow to establish a row header and footer than encloses all of the EmptyItemCell Parts. It determines the number of EmptyItemCells from the list of DataFieldInPattern objects within the **ItemsInPattern** collection on the DynamicPattern, DynamicListView, or DynamicFormView controls.

  If you define any content in the **Template** property and intend to use the EmptyItemCell, add a PlaceHolder control where a list of EmptyItemCells will be inserted and set the **CellPlaceHolderID** property to the ID of that PlaceHolder control.

ASP.NET Markup

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <EmptyItemCell Tag="value" CssClass="value" Style="value"
    NamedStyle="value" NamedStyleFallback="value"
    PatternTemplateName="name" >
    <Template>
      HTML, DynamicLabels, and other controls
    </Template>
    <NamedParts>
      <desDD:EmptyItemCellNamedPatternPart Name="name"
        UseStandardContainer="boolean"
        Same properties as <EmptyItemCell> >
        <Template />
      </desDD:EmptyItemCellNamedPatternPart>
    </NamedParts>
  </EmptyItemCell>
</desDD:DynamicPartsPatterns>
```

Properties on this Part
Click on any of these topics to jump to them:

- Container Tag Properties
- Inner content Properties
- NamedParts Properties
- Behavior Properties

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

Example: EmptyItemCells not used

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <EmptyItem Tag="Div" CssClass="EmptyItem" NamedStyle="EmptyItem" >
    <Template>
      <hr />
    </Template>
  </EmptyItem>
</desDD:DynamicPartsPatterns>
```

Example: EmptyItemCells are used

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <EmptyItem Tag="Tr" CssClass="EmptyItem" NamedStyle="EmptyItem" >
  </EmptyItem>
  <EmptyItemCell Tag="Td" CssClass="EmptyItemCell" NamedStyle="EmptyItemCell" >
    <Template>
      <hr />
    </Template>
  </EmptyItemCell>
</desDD:DynamicPartsPatterns>
```

Example: Both EmptyItem and EmptyItemCells provide content

```
<desDD:DynamicPartsPatterns id="DynamicPartsPattern1" runat="server">
  <EmptyItem Tag="Div" CssClass="EmptyItem" NamedStyle="EmptyItem"
    CellPlaceHolderID="EmptyItemCellPlaceHolder" >
    <Template>
      <br />
      <asp:PlaceHolder id="EmptyItemCellPlaceHolder" runat="server" />
      <br />
    </Template>
  </EmptyItem>
  <EmptyItemCell Tag="Div" CssClass="EmptyItemCell" NamedStyle="EmptyItemCell" >
    <Template>
      <hr />
    </Template>
  </EmptyItemCell>
</desDD:DynamicPartsPatterns>
```

Using this Part within a ListView control

Generally you will use the EmptyItem Part, not the EmptyItemCell Part, from within a ListView control. However, you can use the EmptyItemCell part as shown here.

On a ListView, use it in the **EmptyItemTemplate** property.

```
<EmptyItemTemplate>
  <desDD:DynamicPattern id="DPEIProductName" runat="server"
    PatternTemplateName="name" PartsToGenerate="EmptyItemCell" />
  <desDD:DynamicPattern id="DPEIUnitPrice" runat="server"
    PatternTemplateName="name" PartsToGenerate="EmptyItemCell" />
  <desDD:DynamicPattern id="DPEIDescription" runat="server"
    PatternTemplateName="name" PartsToGenerate="EmptyItemCell" />
</EmptyItemTemplate>
```

Using this Part within a FormView control

Not recommended. There is no Template associated with this Part.

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

Properties on each Part

The Parts use some or all of these properties.

Click on any of these topics to jump to them:

- Container Tag Properties
- Inner content Properties
- NamedParts Properties
- Behavior Properties

Container Tag Properties

- **Tag** (enum PeterBlum.DES.DynamicData.PatternPartContainerTag) – The HTML tag name used to create an opening and closing HTML tag that encloses the inner contents of the Part. The HTML tag also gets values from **Style**, **CssClass**, and **ElementAttributes**. When PatternPartContainerTag.None, the part's inner contents are not enclosed by an HTML tag.

  The enumerated type PatternPartContainerTag has these values:

  - `None` – Do not use a container tag. The inner content is still output unless **Enabled** is false.
  - `Auto` – Determine the tag based on the context. It looks at the Part above to determine the context. If that element is a Table, it sets up a Tr tag. If the element is a Tr tag, it set up a Td or Th tag. If that element is Ol or Ul, it sets up a Li tag.

    A Part is "above" based on its type. Containers are above Rows. Rows are above Cells. Containers are also above GroupHeader, GroupFooter, and NoDataToShow.

    This is the default.
  - `Span`
  - `Div`
  - `Table`
  - `Tr`
  - `Td`
  - `Th`
  - `TableRowWithOneCell` – A Tr tag with a Td tag whose ColSpan attribute makes it generate one cell for the row.
  - `Fieldset`
  - `Ul`
  - `Ol`
  - `Li`
  - `P`
  - `Code`

- **CssClass** (string) – The default style sheet class for the container tag. It defaults to "".

  Can be overridden by a Named Style that supplies its own **CssClass** property value. The Named Style can append its class name to the one here if the **NamedStyle.CssClass** property starts with a + character, like this: **CssClass**="+myclass". It will create a class= like this: `class="defaultclass myclass"`.

CONTINUED ON THE NEXT PAGE

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

- **Style** (System.Web.UI.CssStyleCollection) – A list of cascading style sheet styles in the CSS syntax for the container tag. They are assigned to the HTML tag within the `style=` attribute. When a Named Style is found, its **Style** property will be merged with the Part's **Style** property. When both have the same property name, the Named Style will be used.

  The CSS syntax is this pattern:

  ```
  propertyname:value; propertyname2:value2;
  ```

  For example:

  ```
  vertical-align:bottom; white-space:nowrap
  ```

  The CSS property name and value must be identical to what the browser expects or they will not be used.

  You can locate the CSS properties here: https://developer.mozilla.org/en/CSS_Reference

  When entering the **Style** property in ASP.NET markup, Intellisense is available to assist you in picking style names and defining their values.

  When working programmatically, the Style property can be assigned in two ways:

  To add a single string containing the CSS syntax that replaces the current value of Style, do this:

  ```
  DynamicPattern1.PartType.Style.Value =
      "propertyname:value; propertyname2:value2;"
  ```

  To add a single property, do this:

  ```
  DynamicPattern1.PartType.Add("propertyname", "value")
  ```

- **ElementAttributes** (AttributeCollection) – If you have other HTML tag attributes to display in the HTML tag, like `colspan=` or `valign=`, they are added to this collection. Declare them as follows:

  ASP.NET Markup

  Define them as properties in the Part. There will not be an intellisense or design mode support for these properties. If you are building XHTML pages, use lowercase for your properties.

  ```
  <PartTypePropertyName Tag="Td" CssClass="DataColumn" NamedStyle="DataColumn"
    colspan="2" valign="bottom" >
  ```

  Programmatically adding attributes

  Use the `SetAttribute()` method on the Part property, usually from the `Page_Load()` method. It takes two parameters: style name and value. Both are strings. It will replace the same named property if already added.

  ```
  PartTypePropertyName.SetAttribute("colspan", "2")
  ```

  If you are building XHTML pages, use lowercase for your properties.

- **NamedStyle** (string) – The name from a NamedStyle object in the **NamedStyles** collection of a DynamicPattern, DynamicListView, or DynamicFormView control.

  To lookup a Named Style, specify the same value as in the **Name** property of the NamedStyle object found in the **NamedStyles** collection.

  It defaults to "".

- **NamedStyleFallback** (string) – The name from a NamedStyle object in the **NamedStyles** collection of a DynamicPattern, DynamicListView, or DynamicFormView control. It is used if the Named Style specified in the **NamedStyle** property is not found in the **NamedStyles** collection, this property is used to locate a Named Style.

  To lookup a Named Style, specify the same value as in the **Name** property of the NamedStyle object found in the **NamedStyles** collection.

  It defaults to "".

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

- **Caption** (string) – When **Tag** is `Table`, this is text for an optional `<caption>` tag associated with the `<table>` tag. When **Tag** is `Fieldset`, this is text for an optional `<legend>` tag associated with the `<fieldset>` tag.

  It defaults to "".

- **CaptionCssClass** (string) – The style sheet class name applied to the tag associated with the Caption property. It defaults to "".

- **SelectedCssClass** (string) – *Only on DataRow and DataCell Parts*. When the **DynamicPattern.DataRowAppearance** property is `Selected`, this style sheet class name is used instead of **CssClass** unless this property is unassigned. When using DynamicListView, it sets **DataRowAppearance** when it sets up the **SelectedItemTemplate** for you.

  It defaults to "".

- **AlternativeCssClass** (string) – *Only on DataRow and DataCell Parts*. When the **DynamicPattern.DataRowAppearance** property is `Alternative`, this style sheet class name is used instead of **CssClass** unless this property is unassigned. When using DynamicListView, it sets **DataRowAppearance** when it sets up the **AlternatingItemTemplate** for you.

  It defaults to "".

- **SelectedNamedStyle** and **SelectedNamedStyleFallback** (string) – *Only on DataRow and DataCell Parts*. When the **DynamicPattern.DataRowAppearance** property is `Selected`, these are the NamedStyles used instead of **NamedStyle** and **NamedStyleFallback** unless **SelectedNamedStyle** is unassigned. When using DynamicListView, it sets **DataRowAppearance** when it sets up the **SelectedItemTemplate** for you.

  It defaults to "".

- **AlternativeNamedStyle** and **AlternativeNamedStyleFallback** (string) – *Only on DataRow and DataCell Parts*. When the **DynamicPattern.DataRowAppearance** property is `Alternative`, these are the NamedStyles used instead of **NamedStyle** and **NamedStyleFallback** unless **AlternativeNamedStyle** is unassigned. When using DynamicListView, it sets **DataRowAppearance** when it sets up the **AlternatingItemTemplate** for you.

  It defaults to "".

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

Inner content Properties

Note: None of these properties are available on the Container and GroupContainer Parts.

- **PatternTemplateName** (string) – Gets the content from a Pattern Template file with this name. The ".ascx" extension is optional. When the file is not in the **~/DynamicData/PatternTemplates** folder, specify its URL in the **PatternTemplatesFolderUrl** property.

  The content can come from either or both the **Template** property or a Pattern Template file depending on if this is assigned and the value of **WhenBothAreDefined**.

  It defaults to "".

- **EditPatternTemplateName** (string) – *Only on DataRow and DataCell Parts*. Gets the content from this Pattern Template file with this name when the data entry mode is Edit mode. When unassigned, it falls back to **PatternTemplateFileName**.

  It defaults to "".

- **InsertPatternTemplateName** (string) – *Only on DataRow and DataCell Parts*. Gets the content from this Pattern Template file with this name when the data entry mode is Insert mode. When unassigned, it falls back to **EditPatternTemplateFileName** then to **PatternTemplateFileName**.

  It defaults to "".

- **PatternTemplatesFolderUrl** (string) – The URL to the folder containing the Pattern Template file. If the file is in the **[web application]/DynamicData/PatternTemplates** folder, this property can be unassigned.

  It defaults to "".

- **Template** (ITemplate) – The controls and HTML to output in the inner content of this Part.

  Its behavior depends on the specific implementation of the Parts:

  - HeaderRow - Creates the entire header row's output, ignoring anything defined in the HeaderCell Part, unless you also add a PlaceHolder whose ID is assigned to the **CellsPlaceHolderID** property. If unassigned, it always uses the HeaderCell Part for each DataFieldInPattern supplied.
  - HeaderCell - Usually you add a DynamicColumnTitle control or DynamicLabel. If unassigned, a DynamicColumnTitle is automatically used.
  - DataRow - Creates the entire data row's output, ignoring anything defined in the DataCell Part, unless you also add a PlaceHolder whose ID is assigned to the **CellsPlaceHolderID** property. If unassigned, it always uses the DataCell Part for each DataFieldInPattern supplied.
  - DataCell - Usually you add a DynamicControl and sometimes a DynamicLabel. If unassigned, a DynamicControl is automatically used.
  - FooterRow - Creates the entire Footer row's output, ignoring anything defined in FooterCell, unless you also add a PlaceHolder whose ID is assigned to the **CellsPlaceHolderID** property. If unassigned, it always uses the FooterCell Part for each DataFieldInPattern supplied.
  - FooterCell – Usually you add a DynamicLabel or static HTML. If omitted, a DynamicLabel is automatically used and will be assigned the value from the DataFieldInPattern's or LabelInPattern's **FooterText** property.
  - DataRowSeparator - Creates the entire DataRowSeparator's output, ignoring anything defined in DataRowSeparatorCell, unless you also add a PlaceHolder whose ID is assigned to the **CellsPlaceHolderID** property. If unassigned, it always uses the DataRowSeparatorCell Part.
  - DataRowSeparatorCell – Usually you add a DynamicLabel or static HTML. If omitted, a ` ` symbol is added.
  - EmptyItem - Creates the entire EmptyItem's output, ignoring anything defined in EmptyItemCell, unless you also add a PlaceHolder whose ID is assigned to the **CellsPlaceHolderID** property. If unassigned, it always uses the EmptyItemCell Part for each DataFieldInPattern supplied.
  - EmptyItemCell – Usually you add a DynamicLabel or static HTML. If omitted, a ` ` symbol is added.
  - All other Parts – Do not add any DynamicControls, DynamicLabel, or DynamicColumnTitle controls unless you explicitly assign their DataField property.

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

- **EditTemplate** (ITemplate) – *Only on DataRow and DataCell Parts*. The controls and HTML to output in the inner content of this Part when the data entry mode is Edit mode. When unassigned, it falls back to **Template**.
- **InsertTemplate** (ITemplate) – *Only on DataRow and DataCell Parts*. The controls and HTML to output in the inner content of this Part when the data entry mode is Insert mode. When unassigned, it falls back to **EditTemplate** then to **Template**.
- **WhenBothAreDefined** (enum PeterBlum.DES.DynamicData.TemplatePartOrder) – When both the **Template** and **PatternTemplateName** properties are defined, this determines which to output. One or both can be shown and the order is determined from this property.

  The enumerated type TemplatePartOrder has these values:

  - `TemplateOnly` – Only output the **Template** property. Ignore the Pattern Template
  - `PatternTemplateOnly` – Only output the Pattern Template. Ignore the **Template** property.
  - `PatternTemplateFirst` – Output Pattern Template then **Template**. This is the default.
  - `PatternTemplateLast` – Output **Template** then Pattern Template.
- **CellPlaceHolderID** (string) – *DataRow, HeaderRow, FooterRow, EmptyItem, and DataRowSeparator Parts only* – When using the **Template** to define the overall look of the row, you can still let the DynamicPartsPatterns generate a list of HeaderCell, DataCell, FooterCell, EmptyItemCell, or DataRowSeparatorCell Parts by adding a PlaceHolder control into the Template and assigning its ID here.

  If the ID is not found within the template, an exception is thrown at runtime. If the **PatternTemplateName** property is used, it can contain a PlaceHolder with this ID too, so long as its not inside a nested NamingContainer.

  When unassigned, the Template entirely describes the HTML and controls to output.

  It defaults to "".
- **DefaultHTML** (string) – *HeaderRow and FooterRow Parts only*. When the **Template** and **PatternTemplateName** are not used, default content is generated using this HTML.

  It defaults to ` `.
- **Message** (string) – *NoDataToShow Part only.* The message to display. It can contain HTML. It is used if the **Template** is not setup. It will be shown within the HTML container tag so you can still apply formatting to it through the CssClass and NamedStyle properties.

  It defaults to "".
- **MessageLookupID** (string) – An alternative to **Message**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. Strings must be in the string group Misc.

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

NamedParts Properties

- **NamedParts** (PeterBlum.DES.DynamicData.NamedParts) – *Never in NamedPart objects*. A collection of NamedPart objects that provide alternatives to the output for this Part. Each object *must* be specific to the PartType, such as **ButtonNamedPatternPart** and **DataCellNamedPatternPart**. See "NamedParts Overview".
- **Name** (string) – *Only in NamedPart objects*. A name for this Part. It must be unique against all NamedParts of this Parttype. It is required.
- **UseStandardContainer** (Boolean) – *Only in NamedPart objects*. When `true`, the **Tag**, **Style**, **CssClass** and **ElementAttributes** on this object are used to create the containing tag. When `false`, those properties come from the main part object on DynamicPartsPatterns.

  It defaults to `true`.

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

Behavior Properties

- **Enabled** (Boolean) – When `false`, this part does not generate anything, including a container tag or content. It defaults to `true`.
- **GenerateID** (Boolean) – When `true`, add the id= attribute to the containing tag. The ID is formatting like this: DynamicPartsPatterns.ClientID + "_PT" + (int) PartType.

  The ID is not generated by default because the ID is used by the browser on special cases: when javascript must access the DOM node and when a style sheet needs to be assigned to a specific tag's ID.

  It defaults to `false`.

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

The DynamicStylesTag Control

Use the DynamicStylesTag control on a Pattern Template to apply Named Styles to an HTML tag. Named Styles are defined on Pattern Template files to allow the page developer the ability to customize the formatting of the Pattern Template elements. See “Named Styles: Your pages control the formatting”.

The DynamicStylesTag control lets you specify an HTML tag and default styles, including style sheet class name. It also specifies up to two Named Styles that it supports. (One Named Style is a fallback for when the preferred style is not supplied.) When generating its contents, the DynamicStylesTag control will use the formatting from the Named Style if supplied, and if not, use the default styles.

The Field Templates can also supply their formatting recommendations through their **ContainerTagAttributes** property. The DynamicStylesTag control applies the formatting from **ContainerTagAttributes**. The DynamicStylesTag control does not have to be used in a Pattern Template file to support this feature. (It must be in a Pattern Template file to support Named Styles.)

There are two alternatives to the DynamicStylesTag; these methods defined on the Pattern Template: `GetTagAttributes()` and `ApplyNamedStyle()`.

Click on any of these topics to jump to them:

- Using the DynamicStylesTag Control
- Properties of the DynamicStylesTag control


```
<desDD:DynamicStylesTag runat="server" Tag="Td"
  CssClass="DataColumn" NamedStyle="DataColumn"
  Style="vertical-align:top;" >
    <desDD:DynamicControl ID="DynamicControl1" runat="server" />
</desDD:DynamicStylesTag>
```

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

Using the DynamicStylesTag Control

As you build Pattern Templates, you will add HTML tags to establish format around your DynamicControls and DynamicLabel controls. Consider their `class=` and `style=` attributes. If your tag is `<td>`, also consider `align=` and `valign=` attributes. Will the page developer want to assign these values? If so, the DynamicStylesTag control is appropriate. The page developer will declare NamedStyle objects in the NamedStyles collection of the DynamicPattern, DynamicListView, and DynamicFormView controls.

The general process is as follows:

1. Create the HTML tags and assign their default class, style and other attributes.
2. Define a name for a Named Style to use with the tag.
3. Add that name to the file's documentation and optionally to an XML block of Named Styles.
4. Replace the HTML tag with the DynamicStylesTag control. Here's how to use its properties:
   - **Tag** – The HTML tag name. If assigned, the control will generate the complete HTML tag block: <tagname class= style= other attributes > </tagname>. If unassigned, the control will only generate the attributes that go inside the tag block like this: <tagname *output_goes_here* >. It should be placed within the opening tag that already exists on the page.
   - **CssClass** – The style sheet class to use when the page developer has not supplied a Named Style with a style sheet class.
   - **Style** – Define CSS properties and values that are assigned to the HTML tag within the `style=` attribute.
   - **NamedStyle** – The name of a NamedStyle object that will supply the page developer's overrides to the above formatting properties.
   - **NamedStyleFallback** – An optional second name of a NamedPatternTemplate object that will supply the page developer's overrides to the above formatting properties. When supplied, if the page developer did not supply the NamedStyle object identified by **NamedStyle**, it looks for this named style.
   - **UseContainerTagAttributes** – The Field Templates also may have style recommendations declared in their **ContainerTagAttributes** property. When this is `true` (the default), it uses those recommendations.
5. If you have other HTML tag attributes to assign, like `colspan=` or **`valign=`**, declare them as follows.

ASP.NET Markup

Define them as properties in the DynamicStylesTag. There will not be an intellisense or design mode support for these properties. If you are building XHTML pages, use lowercase for your properties.

```
<desDD:DynamicStylesTag id="DynamicStylesTag1" runat="server" Tag="Td"
  CssClass="DataColumn" NamedStyle="DataColumn"
  colspan="2" valign="bottom" >
```

Programmatically adding attributes

Use the `SetAttribute()` method on the DynamicStylesControl. It takes two parameters: style name and value. Both are strings. It will replace the same named property if already added.

```
DynamicStylesTag1.SetAttribute("colspan", "2")
```

If you are building XHTML pages, use lowercase for your properties.

Creating Named Styles in PatternTemplates

Each type of Pattern Template has specific instructions for creating Named Styles and using this tag.

- DataFieldsPatternTemplate
- PartsPatternTemplate
- DynamicButtonsPatternTemplate
- NoDataFieldsPatternTemplate

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

Properties of the DynamicStylesTag control

The DynamicStylesTag control is subclassed from `System.Web.UI.WebControls.WebControl`.

Click on any of these topics to jump to them:

- Main properties
- Appearance properties
- Behavior properties

Main properties

- **Tag** (enum System.Web.UI.HtmlTextWriterTag) – Specifies the HTML tag to output.

  When a tag name is assigned, it creates the complete opening and closing tags, with class=, style=, and other attributes based on the properties of this control and from a NamedStyle object identified in the **NamedStyles** collection supplied by the page developer. For example, this ASP.NET markup

```
<desDD:DynamicStylesTag runat="server" Tag="Td"
 CssClass="DataColumn" NamedStyle="DataColumn"
 Style="vertical-align:top;" >
   <desDD:DynamicControl ID="DynamicControl1" runat="server" />
</desDD:DynamicStylesTag>
```

  generates this HTML without a matching Named Style:

```
<td class="DataColumn" style="vertical-align:top;" >
    HTML from the DynamicControl's Field Template
</td>
```

  If the page developer adds this NamedStyle

```
<desDD:NamedStyle Name="DataColumn" CssClass="FancyColumn"
 Style="width:200px;" />
```

  it generates this HTML:

```
<td class="FancyColumn" style="vertical-align:top;width:200px" >
    HTML from the DynamicControl's Field Template
</td>
```

  If Tag is `HtmlTextWriterTag.Unknown`, it does not output the begin and end tags, but can still output the attributes string. For example, this ASP.NET markup creates the class= and style= attributes within the text "<tr *attributes* >"

```
<tr <desDD:DynamicStylesTag runat="server" NamedStyle="DataRow"
   CssClass="DataRow" > >
```

  It defaults to `HtmlTextWriterTag.Unknown`.

- **NamedStyle** (string) – The name from a NamedStyle object in the **NamedStyles** collection of a DynamicPattern, DynamicListView, or DynamicFormView control.

  To lookup a Named Style, specify the same value as in the **Name** property of the NamedStyle object found in the **NamedStyles** collection.

  If the DynamicStylesTag control is not within a Pattern Template, this property is ignored.

  It defaults to "".

CONTINUED ON THE NEXT PAGE

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

- **NamedStyleFallback** (string) – The name from a NamedStyle object in the **NamedStyles** collection of a DynamicPattern, DynamicListView, or DynamicFormView control. It is used if the Named Style specified in the **NamedStyle** property is not found in the **NamedStyles** collection, this property is used to locate a Named Style.

  To lookup a Named Style, specify the same value as in the **Name** property of the NamedStyle object found in the **NamedStyles** collection.

  If the DynamicStylesTag control is not within a Pattern Template, this property is ignored.

  It defaults to "".

- **UseContainerTagAttributes** (Boolean) – Determines if the **ContainerTagAttributes** property on each Field Template is applied to these styles.

  This control searches through the Field Templates it contains and adjusts its styles and attributes.

  When there are multiple Field Templates, rules are merged like this:

  - If any have **Wrap**=`false`, set **Wrap** to `false`.
  - The cell alignment of the first set is applied. It ignores those whose Alignment=`NotSet`.
  - Takes the largest width, minwidth, maxwidth, and height. Only compares for the same UnitType. For example, if the first is `Pixels`, it only uses others that are also `Pixels`.

  It defaults to `true`.

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

Appearance properties

These properties provide default class=, style=, and other attributes to the HTML tag.

- **CssClass** (string) – The default style sheet class. It defaults to "".

  Can be overridden by a Named Style that supplies its own **CssClass** property value. The Named Style can append its class name to the one here if the **NamedStyle.CssClass** property starts with a + character, like this: **CssClass**="+myclass". It will create a class= like this: `class="defaultclass myclass"`.

- **Style** (System.Web.UI.CssStyleCollection) – A list of cascading style sheet styles in the CSS syntax. They are assigned to the HTML tag within the `style=` attribute. When a Named Style is found, its **Style** property will be merged with the **DynamicStylesTag.Style** property. When both have the same property name, the Named Style will be used.

  The CSS syntax is this pattern:

  ```
  propertyname:value; propertyname2:value2;
  ```

  For example:

  ```
  vertical-align:bottom; white-space:nowrap
  ```

  The CSS property name and value must be identical to what the browser expects or they will not be used.

  You can locate the CSS properties here: https://developer.mozilla.org/en/CSS_Reference

  When entering the **Style** property in ASP.NET markup, Intellisense is available to assist you in picking style names and defining their values.

  When working programmatically, the Style property can be assigned in two ways:

  To add a single string containing the CSS syntax that replaces the current value of Style, do this:

  ```
  DynamicStylesTag1.Style.Value = "propertyname:value; propertyname2:value2;"
  ```

  To add a single property, do this:

  ```
  DynamicStylesTag1.Style.Add("propertyname", "value")
  ```

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

Behavior properties

- **Visible** (Boolean) – When `false`, this control is not used. It defaults to `true`.

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

DynamicDataManager Control and Adapters

A **DynamicDataManager control** is required on any page that uses ASP.NET Dynamic Data.

The DynamicDataManager control connects your DataBound controls (GridView, DynamicFormView, etc) to the Dynamic Data framework, specifically to the EntityTable and its metadata. For controls that support DynamicField objects, the DynamicDataManager handles scaffolding through either the default or your own FieldGenerator class.

Each type of DataBound control has to communicate with Dynamic Data successfully. For single record controls like DetailsView and FormView, DynamicData needs to know how to get and set the mode with values of ReadOnly, Edit, and Insert. For list controls, it needs to know how to get and set the row currently in edit mode. These are just two examples of a multitude of attributes that the DataBound control shares with Dynamic Data.

DES provides the **DataBound Control Adapter classes** to do this work. You must add one Adapter object for each DataBound control that the DynamicDataManager manages. It has numerous properties to determine how it drives your DataBound control. Here is an example where the DynamicDataManager manages a DynamicFormView to support Insert mode. It is taken from the **Insert.aspx** Page Template file:

```
<desDD:DynamicDataManager ID="DynamicDataManager1" runat="server"
   AutoLoadForeignKeys="true" >
   <Adapters>
      <desDD:DynamicFormViewAdapter DataBoundControlID="RecordDetails"
         ValidationSummaryControlID="ValidationSummary1"
         SupportsEditMode="false" SupportsInsertMode="true"
         CommandsUseRouting="true"
         PostInsertAction="List" PostCancelAction="List" />
   </Adapters>
</desDD:DynamicDataManager>
```

Note that both DES and the ASP.NET Dynamic Data frameworks supply their own DynamicDataManager control. Please use the one from DES (`<desDD:DynamicDataManager>`) instead of the native one (`<asp:DynamicDataManager>`). (DES's control is subclassed from the native one.)

Click on any of these topics to jump to them:

- Features of the DES DynamicDataManager control
- Using the DynamicDataManager control
  - Overview of DataBound Control Adapters
- Properties on DES DynamicDataManager
- Methods on DES DynamicDataManager
  - ChangeMode method
  - GetValidationGroupName method
  - GetDataBoundControlAdapter method
- The BaseDataBoundControlAdapter class
  - Properties common to all DataBound Control Adapters
  - Properties on DetailsViewAdapter, FormViewAdapter, and DynamicFormViewAdapter
- Using Dynamic Data without the normal DataSource controls

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

Features of the DES DynamicDataManager control

There are two versions of the DynamicDataManager control. ASP.NET Dynamic Data supplies `System.Web.DynamicData.DynamicDataManager` and DES Dynamic Data supplies `PeterBlum.DES.DynamicData.DynamicDataManager` (which is a direct subclass). You should use DES's DynamicDataManager control instead of the native control.

DES's DynamicDataManager control has these enhancements:

- Provides a richer FieldsGenerator class. See "Features of the IFieldGeneratorGuidance interfaces".
- Registers controls in the ASP.NET Declarative Syntax. No more calling `RegisterControl()`. *In fact, you never call RegisterControl(), even when doing setup programmatically.*
- Migration of code found on the original Page Templates into DataBound Control Adapter classes, eliminating most of the setup and preventing edits that break some aspect of how the DataBound control works with Dynamic Data. It significantly reduces the code associated with the form.
- Automatic creation of the page-level DynamicValidator control.
- Automatic setup of Validation Groups on Validators, buttons, and ValidationSummary. Also automatically handles server side validation on postback.
- Can customize the actions taken when a command is invoked: use the standard ASP.NET URL Routing, use a specified Url, or use the same control.
- When the web form contains a list and single record view that should work together, simply setting the ConnectedToControlID property will setup all code needed.
- Sets up buttons with features from the Customizer control: DisableOnSubmit, ChangeMonitorEnabled, various confirm messages and more.
- Design mode support

If you elect to use the native DynamicDataManager, you will need use the DataBound Control Adapter classes directly. The sample files in **[DES product folder]/DynamicData/Samples** provide examples using both the DES and native DynamicDataManager.

Recommendation: Use DES's DynamicDataManager control.

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

Using the DynamicDataManager control

Every page that uses ASP.NET Dynamic Data requires a DynamicDataManager with at least one DataBound Control Adapter assigned to its **Adapters** property. There are only a few properties on DynamicDataManager, but there are many properties on each Adapter. The Adapter's properties are very important, determining the various controls on your page that it drives (like the DataBound and ValidationSummary controls), whether your DataBound control supports edit and insert modes, and what page its command buttons (like Edit and Save) show when clicked.

The Page Templates already have their DynamicDataManager controls setup correctly for their use. Here is the DynamicDataManager from **Insert.aspx**:

```
<desDD:DynamicDataManager ID="DynamicDataManager1" runat="server"
   AutoLoadForeignKeys="true" >
   <Adapters>
      <desDD:DynamicFormViewAdapter DataBoundControlID="RecordDetails"
         ValidationSummaryControlID="ValidationSummary1"
         SupportsEditMode="false" SupportsInsertMode="true"
         CommandsUseRouting="true"
         PostInsertAction="List" PostCancelAction="List" />
   </Adapters>
</desDD:DynamicDataManager>
```

Overview of DataBound Control Adapters

DataBound Control Adapters have many features. Here are some broad categories:

- Attaching and handling events on the DataBound control that are specific to Dynamic Data.
- Prepares validation groups on validators and buttons
- Prepares buttons for client-side validation
- Provides server side validation
- Applies the Customizer features to buttons
- Determines what to show when the user invokes a command

Focusing on the `<Adapters>` node (above):

- The class name of the Adapter is associated with the type of DataBound control. **Details.aspx**, **Insert.aspx**, and **Edit.aspx** all use the DynamicFormView, so this uses the DynamicFormViewAdapter. Here are the Adapter types included with DES Dynamic Data:

DataBound control	Adapter class
GridView	GridViewAdapter
DetailsView	DetailsViewAdapter
FormView	FormViewAdapter
ListView	ListViewAdapter
DynamicFormView	DynamicFormViewAdapter
DynamicListView	DynamicListViewAdapter

- Always assign the **DataBoundControlID** property (or **DataBoundControl** property when working programmatically). It refers to the DataBound control.
- If your page has a ValidationSummary control that supports validators within the DataBound control, assign it to the **ValidationSummaryControlID** property (or **ValidationSummaryControl** property when working programmatically.)

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

- Buttons for Edit, Insert and Delete actions are determined by properties on both the DataBound Control Adapter and DataSource control.
    - Edit mode – If the DataBound control either supports editing, or offers an Edit button that uses another DataBound control for editing, set the **SupportsEditMode** property to `true`. DES Dynamic Data determines if the DataBound control supports editing by looking at the DataSource's **EnableUpdate** property. When `true`, it supports editing and an Edit button redraws the same control in edit mode. When `false`, an Edit button locates another DataBound control to handle the editing. (It may do this through ASP.NET URL Routing or other rules that you setup on the DataBound Control Adapter.)
    - Insert mode – If the DataBound control either supports editing, or offers an Insert button that uses another DataBound control for editing, set the **SupportsInsertMode** property to `true`. DES Dynamic Data determines if the DataBound control supports editing by looking at the DataSource's **EnableInsert** property. When `true`, it supports inserting and an Insert button redraws the same control in insert mode. When `false`, an Insert button locates another DataBound control to handle the insertion. (It may do this through ASP.NET URL Routing or other rules that you setup on the DataBound Control Adapter.)
    - Setting the initial mode – This is not controlled by the DataBound Control Adapter. Instead, set the **DefaultMode** property on your DataBound control to `ReadOnly`, `Edit`, or `Insert`.
    - Providing a Delete button – This is not controlled by the DataBound Control Adapter. Set the DataSource's **EnableDelete** property to `true`.
- When any button submits, the DataBound Control Adapter determines what appears next. It could be the same page or another depending on these properties:
    - **CommandsUseRouting** – Follow the ASP.NET URL Routing rules that you setup in the `Application_Start()` method of **Global.asax**. Page Templates always set this to `true`.
    - **PostUpdateAction**, **PostInsertAction**, **PostCancelAction**, and **PostDeleteAction** – Available on the Adapters for single record views, these determine the action for ASP.NET URL Routing after these commands. **PostUpdateAction** applies to the Save button on Edit mode. **PostInsertAction** applies to the Save button on Insert mode. **PostCancelAction** applies to the Cancel button on both Edit and Insert modes. **PostDeleteAction** applies to the Delete button.

      When unassigned, no ASP.NET URL Routing occurs and the user remains on the same form with the DataBound Control in ReadOnly mode.

      When assigned, use one of these strings: "`List`", "`Details`", "`Edit`", or "`Insert`". They are passed to the `MetaTable.GetActionPath()` which uses ASP.NET URL Routing to determine a Url. "`List`" displays the web form used to list records and is the default on **Edit.aspx** and **Insert.aspx** Page Templates. If you prefer them to display **Details.aspx**, change these properties to "`Details`".
    - **DetailsCommandUrl**, **EditCommandUrl**, **InsertCommandUrl** – Replaces ASP.NET URL Routing with your own Url for Delete, Edit, and Insert buttons.

For details on these properties, see "Properties common to all DataBound Control Adapters" and "Properties on DetailsViewAdapter, FormViewAdapter, and DynamicFormViewAdapter".

ALERT: If you create or edit DataBound Control Adapters programmatically, you must do so within the Init event handler of the DynamicDataManager. See examples on the next pages.

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

Properties on DES DynamicDataManager

The DES DynamicDataManager is subclassed from `System.Web.DynamicData.DynamicDataManager`. For properties on that class, see http://msdn.microsoft.com/en-us/library/system.web.dynamicdata.dynamicdatamanager_properties.aspx.

- **AutoLoadForeignKeys** (boolean) – When `true` and using a LinqDataSource, it automatically preloads foreign key data. It has no impact on the features shown to the user. If you leave it `false`, your site will have poorer CPU performance when you are showing foreign key data on your DataBound control. Yet, if you do not show foreign key data at all, it is better to leave this set to `false` as preloading the data is unnecessary work for the CPU.

  Suggested approach: if you are including columns that show foreign keys (usually the **ForeignKey.ascx** and **ForeignKey_Edit.ascx** Field Templates), set it to `true`. Otherwise, leave it `false`.

  It defaults to `false`.

- **Adapters** (PeterBlum.DES.DynamicData.DataBoundControlAdapters) – A list of `PeterBlum.DES.DynamicData.BaseDataBoundControlAdapters` that register the DataBound control and ValidationSummary control to be managed by Dynamic Data. It also provides rules for displaying various buttons and responding to them. See "Using the DynamicDataManager control".

  For properties on these classes, see "The BaseDataBoundControlAdapter class".

  Example: Edit.aspx PageTemplate

```
<desDD:DynamicDataManager ID="DynamicDataManager1" runat="server"
   AutoLoadForeignKeys="true" >
   <Adapters>
      <desDD:DynamicFormViewAdapter DataBoundControlID="RecordDetails"
         ValidationSummaryControlID="ValidationSummary1"
         SupportsEditMode="true" SupportsInsertMode="false"
         CommandsUseRouting="true"
         PostUpdateAction="List" PostCancelAction="List" />
   </Adapters>
</desDD:DynamicDataManager>
```

  Example: Two DataBound controls in ListDetails.aspx

  In this example, assume that there is one ValidationSummary control shared between the two DataBound controls.

```
<desDD:DynamicDataManager ID="DynamicDataManager1" runat="server"
   AutoLoadForeignKeys="true" >
   <Adapters>
      <desDD:DynamicListViewAdapter DataBoundControlID="RecordsList"
         ConnectedToControlID="RecordDetails"
         ValidationSummaryControlID="ValidationSummary1"
         SupportsInsertMode="false" SupportsEditMode="true"
         CommandsUseRouting="true" />
      <desDD:DynamicFormViewAdapter DataBoundControlID="RecordDetails"
         ConnectedToControlID="RecordsList"
         ValidationSummaryControlID="ValidationSummary1"
         SupportsInsertMode="true" SupportsEditMode="true"
         CommandsUseRouting="true" />
   </Adapters>
</desDD:DynamicDataManager>
```

MORE EXAMPLES ON THE NEXT TWO PAGES

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

Example: Creating Adapters programmatically

This example creates the same two Adapters as shown in the previous example. This must be done in the Init event handler of the DynamicDataManager. In this example, it assumes:

```
<desDD:DynamicDataManager OnInit="DynamicDataManager1_Init" [properties] >
```

Note that properties pointing to controls take a direct reference to the control object instead of an ID.

[C#]

```
using PeterBlum.DES.DynamicData;
…
protected void DynamicDataManager1_Init(object pSender, EventArgs e)
{
   DynamicListViewAdapter vListAdapter = new DynamicListView(RecordsList);
   vListAdapter.ConnectedToControl = RecordDetails;
   vListAdapter.ValidationSummaryControl = ValidationSummary1;
   vListAdapter.SupportsInsertMode = false;
   vListAdapter.SupportsEditMode = true;
   vListAdapter.CommandsUseRouting = true;
   DynamicDataManager1.Adapters.Add(vListAdapter);

   DynamicFormViewAdapter vFormAdapter = new DynamicFormView(RecordDetails);
   vFormAdapter.ConnectedToControl = RecordsList;
   vFormAdapter.ValidationSummaryControl = ValidationSummary1;
   vFormAdapter.SupportsInsertMode = true;
   vFormAdapter.SupportsEditMode = true;
   vFormAdapter.CommandsUseRouting = true;
   DynamicDataManager1.Adapters.Add(vFormAdapter);
}
```

[VB]

```
Imports PeterBlum.DES.DynamicData
…
Protected Sub DynamicDataManager1_Init(ByVal pSender As Object,_
   ByVal e As EventArgs)
   Dim vListAdapter As DynamicListViewAdapter = _
      New DynamicListView(RecordsList)
   vListAdapter.ConnectedToControl = RecordDetails
   vListAdapter.ValidationSummaryControl = ValidationSummary1
   vListAdapter.SupportsInsertMode = False
   ListAdapter.SupportsEditMode = True
   ListAdapter.CommandsUseRouting = True
   DynamicDataManager1.Adapters.Add(vListAdapter)

   Dim vFormAdapter As DynamicFormViewAdapter = _
      New DynamicFormView(RecordDetails)
   vFormAdapter.ConnectedToControl = RecordsList
   vFormAdapter.ValidationSummaryControl = ValidationSummary1
   vFormAdapter.SupportsInsertMode = True
   vFormAdapter.SupportsEditMode = True
   vFormAdapter.CommandsUseRouting = True
   DynamicDataManager1.Adapters.Add(vFormAdapter)
End Sub
```

MORE EXAMPLES ON THE NEXT PAGE

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

Example: Editing Adapters programmatically

This example changes the **EditCommandUrl** property on the first DataBound Control Adapter already defined in the DynamicDataManager. This must be done in the Init event handler of the DynamicDataManager. In this example, it assumes:

```
<desDD:DynamicDataManager OnInit="DynamicDataManager1_Init" [properties] >
```

[C#]

```
using PeterBlum.DES.DynamicData;
…
protected void DynamicDataManager1_Init(object pSender, EventArgs e)
{
   DynamicListViewAdapter vListAdapter =
      (DynamicListViewAdapter) DynamicDataManager1.Adapters[0];
   vListAdapter.EditCommandUrl = "~/Registration.aspx";
}
```

[VB]

```
Imports PeterBlum.DES.DynamicData
…
Protected Sub DynamicDataManager1_Init(ByVal pSender As Object,_
   ByVal e As EventArgs)
   Dim vListAdapter As DynamicListViewAdapter = _
      CType(DynamicDataManager1.Adapters(0), DynamicListViewAdapter)
   vListAdapter.EditCommandUrl = "~/Registration.aspx"
End Sub
```

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

Methods on DES DynamicDataManager

Here are methods introduced on the PeterBlum.DES.DynamicData.DynamicManager class. For methods on the ancestor, see http://msdn.microsoft.com/en-us/library/system.web.dynamicdata.dynamicdatamanager_methods.aspx.

Do not use the `RegisterControl()` method. The DataBound Control Adapters call it for you internally.

Click on any of these topics to jump to them:

- ChangeMode method
- GetValidationGroupName method
- GetDataBoundControlAdapter method
- SetupButtonsOnClientSide method
- GetCustomizer method
- RunActionForCommand method
- ExtractValues method

ChangeMode method

Changes the mode of the DataBound Control. It internally calls the `ChangeMode()` method on the DataBound control and performs other setup. Pass the enumerated type `System.Web.UI.WebControls.DataBoundControlMode`.

For example, when implementing an Insert button, use Click event handler to call ChangeMode(DataBoundControlMode.Insert) on the DynamicDataManager instead of calling it directly on the DataBound control.

[C#]

```
public void ChangeMode(Control pDataBoundControl,
  DataBoundControlMode pMode)
```

[VB]

```
Public Sub ChangeMode(ByVal pDataBoundControl As Control,
  ByVal pMode As DataBoundControlMode)
```

Parameters

pDataBoundControl

A reference to the DataBound control whose mode was modified.

pMode

The new mode for the DataBound control: `ReadOnly`, `Edit`, or `Insert`.

Example

See the Samples files:

[C#]

[DES Product Folder]/DynamicData/CS/Samples/Standard Forms on Products Table/FormView.aspx

[VB]

[DES Product Folder]/DynamicData/VB/Samples/Standard Forms on Products Table/FormView.aspx

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

GetValidationGroupName method

Returns the validation group name for a button within the DataBound control passed in. This name is consistent with the validators setup by Dynamic Data.

Use this when DataBinding the Button and your DataBound Control Adapter's **SetupPropertiesOnButtons** property is `false`. (When **SetupPropertiesOnButtons** is `true`, the button's Validation Group name is setup for you.)

[C#]

```
public string GetValidationGroupName(Control pDataBoundControl)
```

```
public string GetValidationGroupName(string pDataBoundControlID)
```

[VB]

```
public Function GetValidationGroupName(ByVal pDataBoundControl As Control)_
  As String
```

```
public Function GetValidationGroupName(ByVal pDataBoundControlID As String)_
  As String
```

Parameters

pDataBoundControl

A reference to the DataBound control whose validation group name is requested.

pDataBoundControlID

The ID to the DataBound control whose validation group name is requested.

Returns

The Validation Group name.

Example

```
<asp:Button id="SaveButton" runat="server"
  CausesValidation="true"
  ValidationGroup="<%# DynamicDataManager1.GetValidationGroupName(FormView1) %>"
  Text="Save" CommandName="Update" />
```

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

GetDataBoundControlAdapter method

Returns the DataBound Control Adapter class associated with the DataBound control specified. Use it to get the Adapter whose properties you want to modify. All modifications must be done in the DynamicDataManager's Init event handler, so call this within the Init event handler.

[C#]

```
public PeterBlum.DES.DynamicData.BaseDataBoundControlAdapter
  GetDataBoundControlAdapter(Control pDataBoundControl)
```

[VB]

```
public Function GetDataBoundControlAdapter(ByVal pDataBoundControl As Control)_
  As PeterBlum.DES.DynamicData.BaseDataBoundControlAdapter
```

Parameters

pDataBoundControl

A reference to the DataBound control whose DataBound Control Adapter is requested.

Returns

A subclass of `PeterBlum.DES.DynamicData.BaseDataBoundControlAdapter` or `null`/`Nothing` if not found.

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

SetupButtonsOnClientSide method

DES can search through a control containing your buttons to setup the Validation Group and apply these properties from the Customizer control: **DisableOnSubmit**, **MayMoveOnClick**, **ChangeMonitorEnables**, **DeleteCommandMessage** (when CommandName="Delete"), **UpdateConfirmMessage** (when CommandName="Insert"), and **CancelConfirmMessage** (when CommandName="Cancel").

In the Page's `OnPreRender()` method, call the `SetupButtonsOnClientSide()` method on the DynamicDataManager control.

ALERT: Always set the **SetupPropertiesOnButtons** property to `false` on the DataBound Control Adapter declared in the DynamicDataManager. Otherwise, you may get excessive (time wasting) searches.

[C#]

```
void SetupButtonsOnClientSide(Control pDataBoundControl,
   Control pContainer,bool pApplyCustomizerRulesToButtons,
   Control pContainerForEnterKeySupport)
```

```
void SetupButtonsOnClientSide(string pDataBoundControlID,
   Control pContainer,
   bool pApplyCustomizerRulesToButtons,
   Control pContainerForEnterKeySupport)
```

[VB]

```
Sub SetupButtonsOnClientSide(ByVal pDataBoundControl As Control,
   ByVal pContainer As Control,
   ByVal pApplyCustomizerRulesToButtons As Boolean,
   ByVal pContainerForEnterKeySupport As Control)
```

```
Sub SetupButtonsOnClientSide(ByVal pDataBoundControlID As String,
   ByVal pContainer As Control,
   ByVal pApplyCustomizerRulesToButtons As Boolean,
   ByVal pContainerForEnterKeySupport As Control)
```

Parameters

pDataBoundControl

Reference to the DataBound control

pDataBoundControlID

ID to the DataBound control

pContainer

The control that contains buttons. It's child control tree will be searched for anything that implements `System.Web.UI.WebControls.IButtonControl`. While you can pass the DataBound Control itself, the search is inefficient. Choose a control that contains these controls as the immediate children.

pApplyCustomizerRulesToButtons

When `true`, apply Customizer rules in addition to validation rules. Leave this `false` if you explicitly set up the Customizer rules using DataBinding on the properties.

pContainerForEnterKeySupport

When assigned, this is a control that contains edit controls and buttons. It will be setup to direct the ENTER key to click a Save button within *pContainer*. When `null/nothing`, the feature is not used.

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

GetCustomizer method

Returns Customizer control associated with the DataBound control passed in.

Use this when DataBinding the Button and your DataBound Control Adapter's **SetupPropertiesOnButtons** property is `false`. (When **SetupPropertiesOnButtons** is `true`, the button's Validation Group name is setup for you.)

[C#]

```
public Customizer GetCustomizer(Control pDataBoundControl)
```

```
public Customizer GetCustomizer(string pDataBoundControlID)
```

[VB]

```
public Function GetCustomizer(ByVal pDataBoundControl As Control)_
  As Customizer
```

```
public Function GetCustomizer(ByVal pDataBoundControlID As String)_
  As Customizer
```

Parameters

pDataBoundControl

A reference to the DataBound control whose validation group name is requested.

pDataBoundControlID

The ID to the DataBound control whose validation group name is requested.

Returns

The Customizer control. It never returns `null`. If no Customizer is associated with the DataBound control, one gets created. This assures the caller that they can always dereference a value from this property instead of testing for `null`/`nothing`.

Example

```
<asp:Button id="SaveButton" runat="server"
  CausesValidation="true"
  DisableOnSubmit="<%# DynamicDataManager1.GetCustomizer(FormView1).DisableOnSubmit %>"
  Text="Save" CommandName="Update" />
```

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

RunActionForCommand method

When using a button that is not contained within the DataBound control, you must setup its Click or Command event handler explicitly to take an action. The action often involves ASP.NET Url Routing, or one of the Command Action Properties found on the DataBound Control Adapter. Instead of writing code to take these actions, call `RunActionForCommand()` within you event handler and let it do the work.

[C#]

```
void RunActionForCommand(Control pDataBoundControl,
   string pCommandName,
   object pDataItem,
   bool pSetMode)
```

```
void RunActionForCommand (string pDataBoundControlID,
   string pCommandName,
   object pDataItem,
   bool pSetMode)
```

[VB]

```
Sub RunActionForCommand(ByVal pDataBoundControl As Control,
   ByVal pCommandName As String,
   ByVal pDataItem As Object,
   ByVal pSetMode As Boolean)
```

```
Sub RunActionForCommand(ByVal pDataBoundControlID As String,
   ByVal pCommandName As String,
   ByVal pDataItem As Object,
   ByVal pSetMode As Boolean)
```

Parameters

pDataBoundControl

Reference to the DataBound control

pDataBoundControlID

ID to the DataBound control

pCommandName

The CommandName property from the button.

pDataItem

Data from databinding that is used to establish querystring parameters that specify which record to show. If `null`, it cannot add those querystring parameters.

Specify it with the Edit, Delete, and Select commands. Pass `null`/`nothing` not used for other commands..

pSetMode

If the command name did not cause an action that redirects, this parameter determines if it should change the mode of the DataBound control. When `true`, it should change the mode. When `false`, no action is taken.

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

ExtractValues method

ExtractValues populates a dictionary with the column name and associated value of all controls within a DataBound control. Normally you don't worry about extracting values. Dynamic Data handles it for you. However, when you use the EnableDynamicData() or DataBoundControlAdapter.EnableDynamicData property, you may be responsible for gathering values from controls yourself. Still, this is not applicable if you do it within the ItemUpdating or RowUpdating events of the DataBound control.

Use this method when you want to extract values outside of ItemUpdating, RowUpdating, or ItemInserted events. The traditional way to write code that gets values from all controls is to use the FindControl() method to retrieve an instance of each control. That's very ineffecient. DES Dynamic Data already has a list of the controls. That's where this method helps. Call it with an empty dictionary to get a list of column names and values.

[C#]

```
public void ExtractValues(Control pDataBoundControl,
  System.Collections.Specialized.IOrderedDictionary pValues)
```

```
public void ExtractValues(string pDataBoundControlID,
  System.Collections.Specialized.IOrderedDictionary pValues)
```

[VB]

```
Public Sub ExtractValues(ByVal pDataBoundControl As Control, _
  ByVal pValues As System.Collections.Specialized.IOrderedDictionary)
```

```
Public Sub ExtractValues(ByVal pDataBoundControlID As String, _
  ByVal pValues As System.Collections.Specialized.IOrderedDictionary)
```

Parameters

pDataBoundControl

A reference to the DataBound control whose control values will be retrieved.

pDataBoundControlID

ID to the DataBound control

pValues

A collection that will be populated with values from all controls. Each element has a key and value. The key is the column name. The value is the value returned from the control in its native type.

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

The BaseDataBoundControlAdapter class

The `PeterBlum.DES.DynamicData.BaseDataBoundControlAdapter` class and its subclasses connect the DataBound control and ValidationSummary to the DynamicDataManager. They let Dynamic Data drive the DataBound control and the controls it contains.

See "Overview of DataBound Control Adapters".

Here are the DataBound Control Adapter classes included with DES Dynamic Data:

DataBound control	Adapter class
DynamicListView	DynamicListViewAdapter
GridView	GridViewAdapter
ListView	ListViewAdapter
DynamicFormView	DynamicFormViewAdapter
DetailsView	DetailsViewAdapter
FormView	FormViewAdapter

ALERT: If you create or edit DataBound Control Adapters programmatically, you must do so within the Init event handler of the DynamicDataManager control.

Click on any of these topics to jump to them:

- Overview of DataBound Control Adapters
- Properties common to all DataBound Control Adapters
  - Connections to Controls on the Page
  - Behavior Properties
  - Automatic Scaffolding Properties
  - Setting up Buttons Properties
  - Command Action Properties
  - Additional Properties
- Properties on DetailsViewAdapter, FormViewAdapter, and DynamicFormViewAdapter
  - Command Action Properties

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

Properties common to all DataBound Control Adapters

Click on any of these topics to jump to them:

- Connections to Controls on the Page
- Behavior Properties
- Automatic Scaffolding Properties
- Setting up Buttons Properties
- Command Action Properties
- Additional Properties

Connections to Controls on the Page

- **DataBoundControlID** (string) - The ID to the DataBound control. It or its companion property, **DataBoundControl**, must be assigned. Always assign a control whose class is compatible with the Adapter, as shown in the above list. If you have created a subclass of the DataBound control, you can still use the Adapter class intended for the ancestor DataBound control.

  This is ID property, not the ClientID or UniqueID. It will be translated into the ClientID at runtime. If the control cannot be found in the current or any parent NamingContainer, an exception is thrown at runtime.

- **DataBoundControl** (Control) – A reference to the DataBound control. This is an alternative to the **DataBoundControlID** property. It must be set programmatically.

- **ValidationSummaryControlID** (string) – The ID to the ValidationSummary control that displays errors for validators within the DataBound control. It is optional. DES Dynamic Data manages the validation group of this ValidationSummary control for you.

  The same ValidationSummary control is allowed on multiple DataBound Control Adapters. It will be giving a list of validation group names for all of the DataBound controls.

  This is ID property, not the ClientID or UniqueID. It will be translated into the ClientID at runtime. If the control cannot be found in the current or any parent NamingContainer, an exception is thrown at runtime.

- **ValidationSummaryControl** (Control) – A reference to the ValidationSummary control. This is an alternative to the **ValidationSummaryID** property. It must be set programmatically.

- **ConnectedToControlID** (string) - The ID to a second DataBound control that is connected to the DataBound control in the DataBoundControlID property. This property is used when List and Detail views are on the same form and interact with the same data. The **ListDetails.aspx** Page Template is an example.

  When **DataBoundControlID** is a List control, **ConnectedToControlID** should be a Single Record control.

  When **DataBoundControlID** is a Single Record control, **ConnectedToControlID** should be a List control.

  When assigned, the List and Single Record DataBound controls have additional event handlers including:

  - Sets a Single Record control to Insert mode when there is nothing in the list.
  - When setting edit mode on one DataBound control, the other control will be set to ReadOnly mode.
  - Sets a Single Record control to ReadOnly mode when the selected item is changed in the List view.

  This is ID property, not the ClientID or UniqueID. It will be translated into the ClientID at runtime. If the control cannot be found in the current or any parent NamingContainer, an exception is thrown at runtime.

- **ConnectedToControl** (Control) – Gets and sets a reference to a second DataBound control that is connected to the **DataBoundControlID**. This is an alternative to the **ConnectedToControlID** property. It must be set programmatically.

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

- **CustomizerControlID** (string) – The ID to the Customizer control that is used by this DataBound Control Adapter. It is optional. When unassigned, the Adapter will search for a Customizer to use. See "Specifying the Customizer on the DataBound Control Adapter".

  The same Customizer control is allowed on multiple DataBound Control Adapters. When searching for a Customizer, it looks at the NamingContainer starting where the DynamicDataManager is located. It looks up the tree's NamingContainer until it finds one. If none is found, the Customizer property will be `null`.

  This is ID property, not the ClientID or UniqueID. It will be translated into the ClientID at runtime. If the control cannot be found in the current or any parent NamingContainer, an exception is thrown at runtime.

- **CustomizerControl** (Control) – A reference to the Customizer control. This is an alternative to the **CustomizerID** property. It must be set programmatically.

- **DynamicValidatorControlID** (string) – The ID to a DES DynamicValidator used to show errors that are thrown from your EntityTable classes. It is optional. When assigned, its validation group is updated to reflect the validators created by DynamicData. When not assigned, a DynamicValidator will be created when **AutoCreateDynamicValidator** is `true` (the default).

  This is ID property, not the ClientID or UniqueID. It will be translated into the ClientID at runtime. If the control cannot be found in the current or any parent NamingContainer, an exception is thrown at runtime.

- **DynamicValidatorControl** (Control) – A reference to the DynamicValidator. This is an alternative to the **DynamicValidatorID** property. It must be set programmatically.

- **EnableDynamicData** (PeterBlum.DES.DynamicData.EnableDynamicDataArgs) – Allows support for non-standard DataSources and when there is no DataSource at all. See "Using Dynamic Data without the normal DataSource controls".

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

Behavior Properties

- **SupportsEditMode** (Boolean) – Set to `true` when the DataBound control either supports editing, or offers an Edit button that uses another DataBound control for editing.

  It defaults to `true`.

  DES Dynamic Data determines if the DataBound control supports editing by looking at the DataSource's **EnableUpdate** property. When **EnableUpdate** is `true`, the DataBound control supports editing and an Edit button redraws the same control in Edit mode. When `false`, an Edit button locates another DataBound control to handle the editing. (It may do this through ASP.NET URL Routing or other properties on the DataBound Control Adapter.)

- **SupportsInsertMode** (Boolean) – Set to `true` when the DataBound control either supports Inserting, or offers an Insert button that uses another DataBound control for Inserting.

  It defaults to `true`.

  DES Dynamic Data determines if the DataBound control supports inserting by looking at the DataSource's **EnableInsert** property. When **EnableInsert** is `true`, the DataBound control supports inserting and an Insert button redraws the same control in Insert mode. When `false`, an Insert button locates another DataBound control to handle the insertion. (It may do this through ASP.NET URL Routing or other properties on the DataBound Control Adapter.)

- **AllowForeignKeyNavigation** (Boolean) – Used by Field Templates that offer hyperlinks to other tables. Determines if they should provide the hyperlink or just a label with the foreign key data.

  ForeignKey.ascx (specifically the `PeterBlum.DES.DynamicData.BaseForeignKeyStaticFTUC` class) uses this.

  It defaults to `true`.

- **SetSelectionFromUrl** (enum PeterBlum.DES.TrueFalseDefault) – For record lists to set the initial selected item based on the URL. When `TrueFalseDefault.True`, if a primary key is found in the route values (typically on the query string), it will get be set as the selected item. This only applies to list controls. It will detect the native ASP.NET list-type controls automatically.

  For any third party control, set it to `TrueFalseDefault.True`. The control must implement `System.Web.UI.WebControls.IPersistSelector` to be supported.

  When set to `TrueFalseDefault.Default` and when IPersistSelector is implemented, it acts like this property is `true`.

  It defaults to `TrueFalseDefault.Default`.

- **ValidationGroupTemplate** (string) – The **Group** property on validators and buttons gets assigned a name automatically by using this string. It is a template that contains tokens to assure the group name is different based on the owning DataControl and the mode.

  Tokens are:

  {ID} - The ID property of the control. When the DataControl is in a NamingContainer that is reused on the page, you must replace the {ID} with something that provides uniqueness for each instance of this adapter. Typically you can just put a lead + character in the template and let DES convert it to the full UniqueID value. Example: "+_{MODE}".

  {MODE} - The Mode, which is "readonly", "edit" and "insert".

  It defaults to "`{ID}_{MODE}`".

- **AutoCreateDynamicValidator** (Boolean) – When the ValidationSummary is assigned but there is no DynamicValidator assigned, this tells the DynamicDataManager to create a DynamicValidator assigned to the DataBound control. The validator will only show its text in the ValidationSummary.

  If you want to control the appearance or position of the DynamicValidator, set this to `false`, create it on the page yourself and assign its ID to **DynamicValidationControlID**.

  It defaults to `true`.

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

Automatic Scaffolding Properties

The FieldGenerator is used with Automatic Scaffolding.

- **FieldsGenerator** (System.Web.UI.IAutoFieldGenerator) – Provides the class that is used for automatic scaffolding.

  Assign it programmatically if you have created your own FieldGenerator. See "Customizing Automatic Scaffolding". *Always assign it in the Init event handler of the DynamicDataManager.*

  When left `null`, it is assigned to DES's own FieldGenerator, `PeterBlum.DES.DynamicData.DESFieldGenerator`.

  These DataBound controls do not support a FieldGenerator: FormView and ListView.

  It defaults to `null`.

- **ScaffoldingSettings** (PeterBlum.DES.DynamicData.DESFieldGeneratorSettings) – This property is a class that hosts rules for Automatic Scaffolding. These same rules can be set when you implement the IFieldGeneratorGuidance interface.

  When used in the ASP.NET markup, declare it as the `<ScaffoldingSettings>` node like this:

```
<desDD:DynamicDataManager ID="DynamicDataManager1" runat="server"
   AutoLoadForeignKeys="true" >
   <Adapters>
      <desDD:DynamicFormViewAdapter DataBoundControlID="RecordDetails"
         ValidationSummaryControlID="ValidationSummary1"
         SupportsEditMode="true" SupportsInsertMode="false" >
         <ScaffoldingSettings Property="value" />
      </desDD:DynamicFormViewAdapter>
   </Adapters>
</desDD:DynamicDataManager>
```

  Here are the properties of the `PeterBlum.DES.DynamicData.DESFieldGeneratorSettings` class:

  - **UseScaffoldColumnAttribute** (boolean) – Determines if columns are added using the ScaffoldColumnAttribute.

    When `true`, columns are added using the rules in the ScaffoldColumnAttribute. Other properties here may eliminate the column, including **ShowChildrenTemplateInEditMode**, **ShowLongStringsInLists**, **ShowReadOnlyColumnsInEditMode**, and **Categories**.

    When `false`, only the `IFieldGeneratorGuidance.CreateField()` method determines if a control is created.

    It defaults to `true`.

  - **UseScaffoldOrdering** (boolean) – The ScaffoldColumnAttribute controls can specify the ordering of fields through the **Position** or **AfterThisColumnName** properties.

    When `true`, any DES.ScaffoldColumnAttribute that specifies ordering through the **Position** or **AfterThisColumnName** properties will have that ordering applied.

    When `false`, they are ignored.

    It defaults to `true`.

  - **ShowLongStringsInLists** (boolean) – A long string is any string column that does not have a maximum length. Since these columns can have very large data, they may create exceptionally large lists of records in the browser. So its a good idea to hide or abbreviate the data.

    When `false`, any column identified as a long string is not added to your page. When `true`, they are.

    It defaults to `false`.

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

- **ShowReadOnlyColumnsInEditMode** (boolean) – In edit or insert mode, it determines if a column that is strictly readonly will appear.

  A column that is strictly readonly is defined by the associated property on the EntityTable class having only a GET clause.

C#	VB

`public decimal TotalPrice`
`{`
`  get { return Price * Quantity; }`
`}`
	`Public Property TotalPrice As Decimal`
	`  Get`
	`  Return Price * Quantity`
	`  End Get`
	`End Property`

  Set to `true` to add when scaffolding. To `false` to omit them.

  It defaults to `false`.

- **ShowChildrenTemplateInEditMode** (boolean) – In edit mode, this determines if the single record style interfaces will show columns of type MetaChildrenColumn (which uses the **Children.ascx** Field Template). **Children.ascx** provides a hyperlink to show another page with a list of records. It may not make sense to offer this within edit mode.

  Set to `true` to display this type of Field Template. To `false` to hide them.

  In native ASP.NET Dynamic Data, it is displayed. So this defaults to `true`.

- **Categories** (string) – A list of category names that must be assigned to the columns to include. Use this to narrow your search. Prior to eliminating a data field by category name, it has already been reviewed for elimination by the ScaffoldColumnAttribute and the other rules of the DESFieldGeneratorSettings class.

  Data fields can have CategoryAttributes in their metadata to assist with setting up Automatic Scaffolding and NamedStyles. CategoryAttributes permit a multiple category names. Just one of those names must match just one name in the Categories property in order for the column to be included or excluded.

  Use a pipe delimited list of the category names when multiple names are needed. Use **ExcludeCategories** to determine if the category is included or excluded.

  It defaults to "".

- **ExcludeCategories** (Boolean) – Determines what happens when the **Categories** property contains a matching name to the CategoryAttribute's value. Normally when the Category name is found in the CategoryAttribute, that column is a match. This property can make that column not match.

  When `true`, exclude columns with any of these category names.

  When `false`, include columns with any of these category names.

  It defaults to `false`.

- **CategoriesWithNoMatchAreExcluded** (Boolean) – Determines what happens when the **Categories** property does *not* contain a matching name to the CategoryAttribute's value, including when the data field lacks a CategoryAttribute.

  When `true`, exclude data fields with any of these category names.

  When `false`, include data fields with any of these category names.

  It defaults to `false`.

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

Setting up Buttons Properties

- **SetupPropertiesOnButtons** (boolean) – Determines if any buttons will be converted to have DES client-side support. It is not used when buttons are managed by a DynamicButtonsPatternTemplate.

  When `true`, the following changes are made to buttons found within the DataBound control:

  - Sets the **CausesValidation** property to `true` when a button's **CommandName** is "`Update`" or "`Insert`". (Does not modify **CausesValidation** on any other button.) These commands represent the Save buttons in Edit and Insert modes.
  - For any button whose **CausesValidation** property is `true`, it sets the appropriate Validation Group name. It will not modify the Validation Group name if the button already has a value assigned.
  - When the Customizer control is available and the **ApplyCustomizerRulesToButtons** property is `true`, the following properties are transferred to each button: **DisableOnSubmit**, **MayMoveOnClick**, **ChangeMonitorEnables**, **DeleteConfirmMessage**, **UpdateConfirmMessage**, and **CancelConfirmMessage**. It will not overwrite an existing **ConfirmMessage** on the button.
  - When there is no Customizer control available or the **ApplyCustomizerRulesToButtons** property is `false`, the Delete button (whose **CommandName** is "`Delete`") is assigned this **ConfirmMessage**: "`Are you sure you want to delete this item?`".

  Additionally, when **SetEnterKeyToSaveButton** is `true`, it will attempt to set the ENTER key to map to the first button whose **CausesValidation** property is `true`. In Edit and Insert mode, that is the Save button. In ReadOnly mode, normally buttons have **CausesValidation** set to `false`.

  When **SetupPropertiesOnButtons** is `false`, you must provide support for these features directly in the control, such as by adding DES Buttons to a FormView or ListView. Leave it set to `true` on GridView and DetailsView controls unless you have replaced the buttons with a TemplateField that contains DES Buttons.

  It defaults to `true`. However, when using a Pattern Template based on PeterBlum.DES.DynamicData.DynamicButtonsPatternTemplate, it handles these actions internally and disables the **SetupPropertiesOnButtons** feature.

- **ApplyCustomizerRulesToButtons** (Boolean) – When `true` and **SetupPropertiesOnButtons** is `true`, the Customizer control's properties for buttons are applied to the buttons contained within the DataBound control.

  Includes these properties: **DisableOnSubmit**, **MayMoveOnClick**, **ChangeMonitorEnables**, **DeleteConfirmMessage**, **UpdateConfirmMessage**, and **CancelConfirmMessage**. It will not overwrite a button whose ConfirmMessage is already assigned.

  Note: Even if there is no Customizer control, the Delete button is assigned to a default confirmation message so long as the button does not have its own ConfirmMessage property assigned.

  This property has no effect when **SetupPropertiesOnButtons** is `false` or there is no Customizer control available.

  It defaults to `true`.

- **SetEnterKeyToSaveButton** (Boolean) – When `true` and **SetupPropertiesOnButtons** is `true`, the Save button will intercept the ENTER key instead of whatever button the browser assigns as the default button.

  The Save button is identified by having the **CausesValidation** property set to `true` or **CommandName** of "`Update`" or "`Insert`". This only applies when the mode is Edit or Insert. In ReadOnly mode, normally buttons have **CausesValidation** set to `false` but if you change one to true, it will support this feature.

  It has no effect when **SetupPropertiesOnButtons** is `false`.

  It defaults to `true`.

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

Command Action Properties

- **CommandsUseRouting** (Boolean) – Determines if Edit, Insert, and Details command buttons use ASP.NET URL Routing to determine a destination page. This is set to `true` on Page Templates.

  Each of these commands has a corresponding property: **EditCommandUrl**, **InsertCommandUrl**, and **DetailsCommandUrl**. If they are assigned, their value determines the URL, not ASP.NET URL Routing.

  When the ***xyz*CommandUrl** property is unassigned and **CommandsUseRouting** is `true`, ASP.NET URL Routing is used.

  When the ***xyz*CommandUrl** property is unassigned and **CommandsUseRouting** is `false`, the command uses the current DataBound control to handle the command.

  It defaults to `false`. (Page Templates have already set this property to true.)

- **EditCommandUrl** (string) – Determines the URL used by the Edit command button to display a page where the record is edited. When assigned, ASP.NET URL Routing is not used for the Edit command. When unassigned, ASP.NET URL Routing is only used when **CommandsUseRouting** is `true`. Otherwise, the command continues to use the same Databound control on the current page.

  Even when the **EditCommandUrl** is used, ASP.NET URL Routing supplies the querystring parameters for the Edit command so you can get a unique record identifier. The `MetaTable.GetActionPath()` method creates a URL with querystring parameters following the rules of ASP.NET URL Routing. Its querystring parameters are moved to the end of the Url specified in the **EditCommandUrl** property. They reflect the primary key field names and values. For example “CategoryID=3535” for the primary key “CategoryID” in the Category table and value from that column “3535”.

  Your destination page must use these querystring parameters to select the correct record.

  The token "~" is supported as the web application root folder. You can use a relative path. For example, "Edit.aspx" will use the current path (such as /Category/) and append the file name.

  It defaults to "".

- **InsertCommandUrl** (string) – Determines the URL used by the Insert command button to display a page where a new record is entered. When assigned, ASP.NET URL Routing is not used for the Insert command. When unassigned, ASP.NET URL Routing is only used when **CommandsUseRouting** is `true`. Otherwise, the command continues to use the same Databound control on the current page.

  The token "~" is supported as the web application root folder. You can use a relative path. For example, "Insert.aspx" will use the current path (such as /Category/) and append the file name.

  It defaults to "".

- **DetailsCommandUrl** (string) – Determines the URL used by the Details command button to display a page where the record is shown. When assigned, ASP.NET URL Routing is not used for the Details command. When unassigned, ASP.NET URL Routing is only used when **CommandsUseRouting** is `true`. Otherwise, the command continues to use the same Databound control on the current page.

  Even when the **DetailsCommandUrl** is used, ASP.NET URL Routing supplies the querystring parameters for the Details command so you can get a unique record identifier. The `MetaTable.GetActionPath()` method creates a URL with querystring parameters following the rules of ASP.NET URL Routing. Its querystring parameters are moved to the end of the Url specified in the **DetailsCommandUrl** property. They reflect the primary key field names and values. For example “CategoryID=3535” for the primary key “CategoryID” in the Category table and value from that column “3535”.

  Your destination page must use these querystring parameters to select the correct record. The token "~" is supported as the web application root folder.

  You can use a relative path. For example, "Details.aspx" will use the current path (such as /Category/) and append the file name.

  It defaults to "".

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

Additional Properties

These are mostly read-only properties found on all DataBound Control Adapter classes. Since the Adapter is available to Field Templates and Pattern Templates, you may use it as you write code for those tools.

- **DynamicDataManager** (DynamicDataManager) – Gets a reference to the DynamicDataManager control that contains this DataBound Control Adapter.
- **DynamicDataSource** (System.Web.UI.IDynamicDataSource) – Gets a reference to the DataSource used by this DataBound Control Adapter. The IDynamicDataSource interface provides easy access to the **EnableUpdate**, **EnableInsert**, and **EnableDelete** properties.
- **DataBoundControlType** (enum PeterBlum.DES.DynamicData.DataBoundControlType) – Identifies if the DataBound control is a List or single record oriented control.

  The enumerated type `PeterBlum.DES.DynamicData.DataBoundControlType` has these values:

  - List
  - Single
- **MetaTable** (System.Web.DynamicData.MetaTable) – Gets the MetaTable used by the DataSource.
- **Customizer** (PeterBlum.DES.DynamicData.Customizer) – Gets a reference to the Customizer control associated with this DataBound Control Adapter. It returns a default Customizer control if none is found elsewhere. A default Customizer control is an instance of Customizer with all properties at its default. This assures the caller that they can always dereference a value from this property instead of testing for `null`.

  While the **CustomizerControl** property may contain a reference to the same control, it is only setup if the user explicitly assigns the Customizer to the DataBound Control Adapter. The **Customizer** property gives the Customizer control even if it has to search the page for that control. Of the two, the **Customizer** property is recommended for getting to this control within your Field Templates and Pattern Templates.
- **DESPage** (PeterBlum.DES.DESPage) – A local reference to the **PeterBlum.DES.Global.Page** property.

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

Properties on DetailsViewAdapter, FormViewAdapter, and DynamicFormViewAdapter

These properties are in addition to those defined in "Properties common to all DataBound Control Adapters".

Command Action Properties

- **PostUpdateAction** (string) – When setup, this is a ASP.NET URL Routing action that will be automatically run after an Update command is successful. The Update command occurs on Edit mode when the user elects to save.

  When assigned, it must be a command that is supported by your ASP.NET URL Routing setup. Typical actions commands are defined in `System.Web.DynamicData.PageAction`, including `"Edit"`, `"Insert"`, `"Details"` and `"List"`. It is passed to the `MetaTable.GetActionPath()` method to create the URL. For examples, see the **Edit.aspx** and **Insert.aspx** Page Templates, both which set this to "`List`".

  If unassigned, no ASP.NET URL Routing action is taken automatically. That leaves the user on the current form, which is often correct because a single record DataBound control can handle ReadOnly, Edit, and Insert modes.

  It defaults to "".

- **PostInsertAction** (string) – When setup, this is a ASP.NET URL Routing action that will be automatically run after an Insert command is successful. The Insert command occurs on Insert mode when the user elects to save.

  When assigned, it must be a command that is supported by your ASP.NET URL Routing setup. Typical actions commands are defined in `System.Web.DynamicData.PageAction`, including `"Edit"`, `"Insert"`, `"Details"` and `"List"`. It is passed to the `MetaTable.GetActionPath()` method to create the URL. For examples, see the **Edit.aspx** and **Insert.aspx** Page Templates, both which set this to "`List`".

  If unassigned, no ASP.NET URL Routing action is taken automatically. That leaves the user on the current form, which is often correct because a single record DataBound control can handle ReadOnly, Edit, and Insert modes.

  It defaults to "".

- **PostCancelAction** (string) – When setup, this is a ASP.NET URL Routing action that will be automatically run after an Cancel command is successful. The Cancel command occurs on Edit and Insert modes when the user elects to cancel.

  When assigned, it must be a command that is supported by your ASP.NET URL Routing setup. Typical actions commands are defined in `System.Web.DynamicData.PageAction`, including `"Edit"`, `"Insert"`, `"Details"` and `"List"`. It is passed to the `MetaTable.GetActionPath()` method to create the URL. For examples, see the **Edit.aspx** and **Insert.aspx** Page Templates, both which set this to "`List`".

  If unassigned, no ASP.NET URL Routing action is taken automatically. That leaves the user on the current form, which is often correct because a single record DataBound control can handle ReadOnly, Edit, and Insert modes.

  It defaults to "".

- **PostDeleteAction** (string) – When setup, this is a ASP.NET URL Routing action that will be automatically run after a Delete command is successful.

  When assigned, it must be a command that is supported by your ASP.NET URL Routing setup. Typical actions commands are defined in `System.Web.DynamicData.PageAction`, including `"Edit"`, `"Insert"`, `"Details"` and `"List"`. It is passed to the `MetaTable.GetActionPath()` method to create the URL.

  If unassigned, no ASP.NET URL Routing action is taken automatically. That leaves the user on the current form, which is often correct because a single record DataBound control can handle ReadOnly, Edit, and Insert modes.

  It defaults to "".

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

Useful Methods on DataBound Control Adapters

The following methods are useful when writing Field Templates and Pattern Templates.

Click on any of these topics to jump to them:

- FindByControl static method
- ChangeMode method
- GetMode method
- SetupButtonsOnClientSide method
- GetValidationGroupName method

FindByControl static method

Returns the DataBound Control Adapter object that manages the control passed in. The control can be the actual DataBound control or any control inside the DataBound control.

[C#]

```
static public BaseDataBoundControlAdapter FindByControl(Control pControl)
```

[VB]

```
Shared Public Function FindByControl(ByVal pControl As Control) _
      As BaseDataBoundControlAdapter
```

Parameters

pControl

The control whose DataBound Control Adapter that you want.

Returns

The DataBound Control Adapter or `null`/`nothing` if not found.

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

ChangeMode method

Call to change the mode of the DataBound control. There is a similar method on the DynamicDataManager that does the same thing (internally calling this method). Use this method if you have direct access to the DataBound Control Adapter.

This internally calls the `ChangeMode()` method of the actual DataBound control and performs setup required by the Adapter.

[C#]

```
public void ChangeMode(DataBoundControlMode pMode)
```

[VB]

```
Public Sub ChangeMode(ByVal pMode As DataBoundControlMode)
```

Parameters

pMode

The mode: ReadOnly, Edit, or Insert.

GetMode method

Returns the current mode of the DataBound control.

[C#]

```
public DataBoundControlMode GetMode()
```

[VB]

```
Public Function GetMode() As DataBoundControlMode
```

Returns

The current mode of the DataBound control.

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

SetupButtonsOnClientSide method

Looks through the buttons of the Container to setup client-side validation, Customizer rules, the button which ENTER clicks.

Use this after setting **SetupPropertiesOnButtons** to `false` to update buttons for the client-side within a specific container control. It will find all buttons and create client side support for them.

It should be called in `OnPreRender()`.

[C#]

```
public void SetupButtonsOnClientSide(Control pContainer,
      DataBoundControlMode pMode,
      bool pApplyCustomizerRulesToButtons,
      Control pContainerForEnterKeySupport)
```

[VB]

```
Public Sub SetupButtonsOnClientSide(ByVal pContainer As Control,
      ByVal pMode As DataBoundControlMode,
      ByVal pApplyCustomizerRulesToButtons As Boolean,
      ByVal pContainerForEnterKeySupport As Control)
```

Parameters

pContainer

The control that contains buttons. It's child control tree will be searched for anything that implements IButtonControl.

pMode

The Mode of the DataBound Control. This is used to define the validation group name. It can be retrieved from the `GetMode()` method.

pApplyCustomizerRulesToButtons

When `true`, apply Customizer rules in addition to validation rules.

pContainerForEnterKeySupport

When assigned, this is a control that contains edit controls and buttons. It will be setup to direct the ENTER key invoked in any edit control to click a Save button within *pContainer*. When `null`, the feature is not used.

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

GetValidationGroupName method

Returns the validation group name to apply to any control within the DataBound control. Normally you will not use this on validators because their Field Templates will set them up. This is useful for buttons when the **SetupPropertiesOnButtons** property is `false`.

[C#]

```
public string GetValidationGroupName(DataBoundControlMode pMode)
```

[VB]

```
Public Function GetValidationGroupName(ByVal pMode As DataBoundControlMode) _
      As String
```

Parameters

pMode

The mode for to be validated. Use the `GetMode()` method to get the current mode.

Returns

The Validation Group name to assign to the **Group** property on buttons and validators.

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

Using Dynamic Data without the normal DataSource controls

The DataSource control is one of the key elements when building a web form that uses Dynamic Data. In particular, the DataSource control must implement the `System.Web.DynamicData.IDynamicDataSource` interface. Such functionality is implemented in the LinqDataSource, EntityDataSource, DomainDataSource, EntityDAODataSource, and POCODataSource controls.

When using ASP.NET 4 or higher, several other solutions can be used with Dynamic Data:

- ObjectDataSource control
- A Entity class that is not defined by LINQ to SQL or the ADO.NET Entity Framework
- A System.Data.DataTable object created by a query with ADO.NET

Before going into detail, the solutions here are a fallback, in case you have existing code that you want to apply to Dynamic Data. In fact, if you use the controls from Versatile DataSources, you can handle all of the above cases with more functionality and easier setup. **You are strongly recommended to use Versatile DataSources.**

Click on any of these topics to jump to them:

- The ObjectDataSource control
- An Entity class
- A DataTable

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

The ObjectDataSource control

Note: Requires ASP.NET 4 or higher

With the ObjectDataSource control, you can define an Entity class to describe your data and a Data Access Object to perform the CRUD ("create, read, update, delete") operations.

When using this DataSource, set it up as documented: ObjectDataSource Web Server Control Overview

Add the business attributes to the Entity class. See "Defining business logic on each EntityTable".

The DynamicDataManager control recognizes the ObjectDataSource control and installs extensions to add a System.Web.DynamicData.MetaTable class that describes your Entity type.

For examples (in C# only), see

[DES product folder]\DynamicData\CS\Samples\Non standard DataSource used\ObjectDataSource.aspx

[DES product folder]\DynamicData\CS\Samples\Non standard DataSource used\ ObjectDataSource_InsertMode.aspx

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

An Entity class

Note: Requires ASP.NET 4 or higher

Suppose you have created an Entity class that is not part of LINQ to SQL or ADO.NET Entity Framework. Normally you could connect it to the ObjectDataSource, but that requires defining a Data Access Object with CRUD methods.

Instead, you can use the `EnableDynamicData()` method on the DynamicDataManager control or the **Adapter.EnableDynamicData** property on the DynamicDataManager control. These two features install extensions to add a System.Web.DynamicData.MetaTable class that describes your Entity type.

The EnableDynamicData() method is just an easy way to set the **Adapter.EnableDynamicData** property. Use the method when working programmatically.

```
private void DynamicDataManager_Init(object sender, EventArgs e)
{
  DynamicDataManager1.EnableDynamicData(GridView1, typeof(BLL.SampleEntity));
}
```

Use the property when working in the ASP.NET markup.

```
<desDD:DynamicDataManager ID="DynamicDataManager" runat="server"  >
   <Adapters>
      <desDD:GridViewAdapter DataBoundControlID="GridView1"
         SupportsInsertMode="False" SupportsEditMode="false"
         EnableDynamicData-EntityTypeName="BLL.SampleEntity" />
   </Adapters>
</desDD:DynamicDataManager>
```

For examples (in C# only), see

[DES product folder]\DynamicData\CS\Samples\Non standard DataSource used\ EnableDynamicData_EntityType.aspx

[DES product folder]\DynamicData\CS\Samples\Non standard DataSource used\ EnableDynamicData_DataContext.aspx

The EnableDynamicData method

This overloaded method is declared on the DynamicDataManager control. Always call it from within the Init event handler of the DynamicDataManager control.

[C#]

```
void EnableDynamicData(Control pDataBoundControl, Type pEntityType)

void EnableDynamicData(Control pDataBoundControl, Type pEntityType,
   object pDefaultEntityForInsertMode)

void EnableDynamicData(Control pDataBoundControl,
   PeterBlum.DES.DynamicData.EnableDynamicDataArgs pArgs)
```

[VB]

```
Sub EnableDynamicData(ByVal pDataBoundControl As Control, _
   ByVal pEntityType As Type)

Sub EnableDynamicData(ByVal pDataBoundControl As Control, _
   ByVal pEntityType As Type, _
   ByVal pDefaultEntityForInsertMode As Object)

Sub EnableDynamicData(ByVal pDataBoundControl As Control, _
   ByVal pArgs As PeterBlum.DES.DynamicData.EnableDynamicDataArgs)
```

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

Parameters

pDataBoundControl

A reference to the Databound control, such as GridView, ListView, DetailsView, and FormView. This first parameter also can be a string with the ID of the Databound control. The Databound control and DynamicDataManager must be in the same naming container when passing a string.

pEntityType

The type for the Entity class.

pDefaultEntityForInsertMode

When the Databound control is in Insert mode, this is an instance of the Entity class that is used to initialize the fields, establishing default values for new records.

pArgs

All available properties to enable Dynamic Data. See “”.

The BaseDataControlAdapter.EnableDynamicData property

The subclasses of the BaseDataControlAdapter class are used to associate a DataBound control with various rules. It provides the **EnableDynamicData** property to enable Dynamic Data in non-standard situations. You can set this property using the `DynamicDataManager.EnableDynamicData()` method, above, or edit it programmatically.

The **EnableDynamicData** property uses the class PeterBlum.DES.DynamicData.EnableDynamicDataArgs. Here are its properties.

- **EntityTypeName** (string) – It provides the name of a type that represents the Entity class. If you are working programmatically, it is better to use the **EntityType** property. This property is used in ASP.NET markup.

  You can use the ObjectDataSource control without using this property. Instead assign the same value in the **ObjectDataSource.DataObjectTypeName** property.

  The type must be compatible with `System.Web.Compilation.BuildManager.GetType()`, which generally demands "top level assemblies", like the **App_Code** folder, global.asax and those that are explicitly included within the Compilation section of the **web.config** file.

- **EntityType** (Type) – Provides the type for the Entity class. It is an alternative to **EntityTypeName** that should be used when working programmatically.

  You can use the ObjectDataSource control without using this property. Instead assign the same value in the **ObjectDataSource.DataObjectTypeName** property.

- **DataContextTypeName** (string) – It provides the name of a type that represents the DataContext class. If you are working programmatically, it is better to use the **DataContextType** property. This property is used in ASP.NET markup.

  It is only used when the MetaModel was registered with the DataContext class that is associated with your Entity class.

- **DataContextType** (Type) – Provides the type for the DataContext class. It is an alternative to **DataContextTypeName** that should be used when working programmatically.

  It is only used when the MetaModel was registered with the DataContext class that is associated with your Entity class.

- **EntityCustomTypeDescriptor** (CustomTypeDescriptor) – A CustomTypeDescriptor class can give the appearance of an Entity class from non-class data, like the `System.Data.DataTable`, which describes the columns and data of a query. This property is generally used by the `DynamicDataManager.EnableDynamicData(DataTable)` method as it establishes the `PeterBlum.DES.DynamicData.DataTableCustomTypeDescriptor` here.

- **UseDESFieldTemplateFactory** (Boolean) – When the MetaTable is created, ASP.NET Dynamic Data sets up a default Field Template Factory class, System.Web.DynamicData.SimpleFieldTemplateFactory. DES has a more powerful Field Template Factory class. When this is true, it switches to DES's more powerful class. It defaults to true.

- **DefaultEntityForInsertMode** (Object) – When the Databound control is in Insert mode, this is an instance of the Entity class that is used to initialize the fields, establishing default values for new records.

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

A DataTable

The System.Data.DataTable class is a mainstay of ADO.NET programming. It represents the results of a query, describing both the structure of the columns and the rows of data. Dynamic Data normally works from a different structure of the columns, the "Entity class" and its metadata for business logic. Yet you can still use the DataTable with some conditions:

- It is a read-only system. Do not use it for edit or insert mode as there is no mechanism to convert the onscreen data into ADO.NET CRUD actions.
- You can attach business logic to a DataTable, giving each column the same Attributes you would to an Entity class.

Here's how to use a DataTable with Dynamic Data. Do all of the following in the Init event handler of the DynamicDataManager control.

6. Use ADO.NET to query your database and prepare a DataTable object.
7. For any desired business rules, create instances of the attributes and assign them to the DataColumn.ExpandedProperties collection. The collection takes a name and value. Aside from the name being unique, its value does not matter. The second element, value, is the attribute.
8. Pass the DataTable to the `DynamicDataManager.EnableDynamicData()` method.

Example

```
private void DynamicDataManager_Init(object sender, EventArgs e)
{
   // Get DataTable as a list of all Products
   string connectionString =
     ConfigurationManager.ConnectionStrings["Northwind"].ConnectionString;
  using (SqlConnection conn = new SqlConnection(connectionString)) {
    conn.Open();
    SqlCommand command = new SqlCommand("SELECT * FROM Products", conn);
    SqlDataAdapter adapter = new SqlDataAdapter(command);
    dataTable = new DataTable("Products");
    adapter.Fill(dataTable);
  }

   // Add some metadata
   PropertyCollection productNameMd =
     dataTable.Columns["ProductName"].ExtendedProperties;
   productNameMd["DisplayName"] = new DisplayNameAttribute("Name");
   PropertyCollection UnitPriceMd =
     dataTable.Columns["UnitPrice"].ExtendedProperties;
   productNameMd["UnitPrice"] = new DataTypeAttribute(DataType.Currency);

   // Connect with the DynamicDataManager
   DynamicDataManager1.EnableDynamicData(GridView1, dataTable);
}
DataTable dataTable;
```

For an example (in C# only), see

[DES product folder]\DynamicData\CS\Samples\Non standard DataSource used\ EnableDynamicData_DataTable.aspx

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

DES.DynamicControl: The heart of Dynamic Data

The DynamicControl is a placeholder for where a Field Template will generate its content. To work, a DynamicControl must know the data field name, which you set in its **DataField** property. From there, it can select the correct Field Template and set it up. You can override the default choice for the Field Template (which is based on attributes on the data field's metadata) with the **UIHint** property.

There are actually two DynamicControl classes available when using DES Dynamic Data.
`System.Web.DynamicData.DynamicControl` is native to ASP.NET Dynamic Data.
`PeterBlum.DES.DynamicData.DynamicControl` is a subclass of the native control introduced in DES Dynamic Data. It provides additional features and is the recommended control class. This section describes DES' DynamicControl.

Click on any of these topics to jump to them:

- Features of DES.DynamicControl
- Using DES.DynamicControl
- Adding a DES.DynamicControl
- Properties on DES.DynamicControl

Features of DES.DynamicControl

- Supply alternative values for properties on the Field Template to customize its behavior, making Field Templates very flexible.
- Using the DES.FieldTemplateFactory or the value of a UIHint, it loads a Field Template file and generates its content

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

Using DES.DynamicControl

The DynamicControl has several tasks:

- Identify the data field whose data will be viewed or edited. See "Assigning the DataField"
- Select a Field Template, using the DES.FieldTemplateFactory or the value of a UIHint. See "Selecting the Field Template".
- Supply alternative values for properties on the Field Template to customize its behavior. See "Supply alternative values for properties on the Field Template".

Click on any of these topics to jump to them:

- Where to add DynamicControls
- Assigning the DataField
- Selecting the Field Template
- Supply alternative values for properties on the Field Template
- See also "How to: Use ASP.NET Dynamic Data in Templated Data-Bound Controls" (an article)

Where to add DynamicControls

DynamicControl controls display Field Templates anywhere ASP.NET web controls are permitted within a DataBound control setup for Dynamic Data:

- The `<xyzTemplate>` properties of FormView and DynamicFormView controls
- The `<xyzTemplate>` properties of ListView and DynamicListView controls
- TemplateField object within a GridView or DetailsView control
- Pattern Templates
- Entity Templates (a feature of Dynamic Data v4)
- The TemplateNotInPattern object used with the DynamicPattern, DynamicFormView, and DynamicListView controls

Assigning the DataField

The **DataField** property must be assigned in order for DynamicControl to do its job. There are two situations to consider.

When the DynamicControl is contained within Pattern Templates or Entity Templates, do not assign it yourself. That's the job of the DynamicPattern and DynamicEntity controls.

In all other cases, assign it. Use the name of the property, not any name specified on a DisplayNameAttribute assigned to that column.

```
<desDD:DynamicControl id="DC1" runat="server" DataField="ColumnName" />
```

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

Selecting the Field Template

Dynamic Data knows exactly which Field Template to use for your data field because you have established that information in the business logic, with the field's actual data type, with the DataTypeAttribute or with the UIHintAttribute.

Sometimes you want to override that choice on the DynamicControl. Use the **UIHint** property. Specify the name of the Field Template, omitting the extensions ".ascx", "_edit.ascx", and "_insert.ascx".

```
<desDD:DynamicControl id="DC1" runat="server"
    DataField="ColumnName" UIHint="TemplateName" />
```

The DynamicControl needs to know the data entry mode so it can select between *Field Template*.ascx, *Field Template*_Edit.ascx, and *Field Template*_Insert.ascx. Use the **Mode** property. It defaults to ReadOnly, so only set it if the mode is Insert or Edit.

```
<desDD:DynamicControl id="DC1" runat="server"
    DataField="ColumnName" Mode="Edit" />
```

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

Supply alternative values for properties on the Field Template

ALERT: This feature is only supported by the DES.DynamicControl, not the native one.

Each Field Template is derived from a class, and that class offers public properties to customize its behavior. For example, the base class used by all DES Field Templates, `PeterBlum.DES.DynamicData.DESFieldTemplateUserControl`, has these public properties: **DataControlWidth**, **DataControlHeight**, **UseDescriptionAsToolTip**, **UseDependencyAttribute**, and **ValueWhenBlank**. (See "Field Template classes and properties".)

The DES.DynamicControl has several properties where you can define a property list for the Field Templates. They are:

- **DefaultPropertyList** – A property list that is applied the Field Template regardless of the mode. See below for the syntax. The mode-specific PropertyLists can override identical properties found here. Any unique properties declared here will still be set. This allows you to setup properties common to all modes here.
- **ReadOnlyPropertyList** – A property list that is applied to a Field Template in ReadOnly mode. See below for the syntax. This property list is applied after **DefaultPropertyList**, allowing individual properties to override values already applied.
- **EditPropertyList** – A property list that is applied to a Field Template in Edit mode. See below for the syntax. This property list is applied after **DefaultPropertyList**, allowing individual properties to override values already applied.
- **InsertPropertyList** – A property list that is applied to a Field Template in Insert mode. See below for the syntax. This property list is applied after **DefaultPropertyList**, allowing individual properties to override values already applied.

Syntax of these properties

The PropertyList properties use the ASP.NET Declarative Syntax, which means it looks like properties you would type into an `<asp:class here>` tag. However, you must enclose every value in single quotes.

Here's the overall pattern:

```
DefaultPropertyList="PropertyName1='propertyvalue' PropertyName2='propertyvalue'"
```

Hint: If you need single quotes within your property values, enter them as "&#39".

```
<desDD:DynamicControl id="DC1" runat="server" DataField="ColumnName"
      InsertPropertyList="DataControlWidth='40px' ValueWhenBlank='FillMe'" />
```

The property value is a string that must be compatible with the TypeConverter for the property's datatype. For example, a DateTime property uses the DateTimeTypeConverter. It converts strings using the ShortDatePattern in the current thread's CultureInfo object or using the culture neutral format "yyyy-MM-dd".

You can refer to the child properties of properties using the syntax: `PropertyName-ChildPropertyName='value'`. This is important because you have access to the data control through the **DataControl** property and the ColumnValidatorManager through the **ColumnValidatorManager** property.

For example, `InsertPropertyList="DataControl-BackColor='aliceblue'"`.

When you declare an unknown property name, it will be ignored. If you want to see any errors caused by incorrect property names or values, set `<% @Page Trace="true" %>` and view the trace.

Please refer to "Field Template classes and properties" to learn about the available properties.

Examples

```
<desDD:DynamicControl id="DC1" runat="server" DataField="ColumnName"
      DefaultPropertyList="DataControlWidth='100px'"
      InsertPropertyList="DataControlWidth='80px' ValueWhenBlank='FillMe'" />
```

```
<desDD:DynamicControl id="DC1" runat="server" DataField="ColumnName"
      DefaultPropertyList="UseDescriptionAsToolTip='false'" />
```

```
<desDD:DynamicControl id="DC1" runat="server" DataField="ColumnName"
      InsertPropertyList="DataControl-BackColor='aliceblue'" />
```

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

When creating DES.DynamicControl programmatically

When you create or modify a DynamicControl object programmatically, don't assign values to the above properties because those properties require parsing and conversion to their native types. That uses extra CPU time that can be avoided by using the following technique.

The DynamicControl class defines these properties to be used programmatically: **DefaultProperties**, **ReadOnlyProperties**, **EditProperties**, and **InsertProperties**. Each of these are a collection which take a property name (as a case sensitive string) and a value (in the actual type expected by the property, like an integer or boolean).

Call the method `AddAsDefault()` to add to the **DefaultProperties** collection. Call the method `AddWithMode()` to add to any of the other properties.

[C#]

```
void AddAsDefault(string pPropertyName, object pValue)

void AddWithMode(System.Web.UI.WebControls.DataBoundControlMode pMode,
     string pPropertyName, object pValue)
```

[VB]

```
Sub AddAsDefault(ByVal pPropertyName As String, ByVal pValue As Object)

Sub AddWithMode(ByVal pMode As System.Web.UI.WebControls.DataBoundControlMode, _
     ByVal pPropertyName As String, ByVal pValue As Object)
```

Parameters

pPropertyName

The name of the property on the Field Template. It must be a case sensitive match.

pValue

The value to pass to that property in the actual type of the property.

pMode

Selects one of these properties to hold the property value: **ReadOnlyProperties**, **EditProperties**, and **InsertProperties**.

Example

[C#]

```
using DES=PeterBlum.DES.DynamicData;
…
DES.DynamicControl vDynamicControl = new DES.DynamicControl();
vDynamicControl.AddAsDefault("DataControlWidth", new Unit(100, UnitType.Pixels));
vDynamicControl.AddWithMode("DataControl-BackColor", "aliceblue");
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
…
Dim vDynamicControl As DES.DynamicControl = New DES.DynamicControl()
vDynamicControl.AddAsDefault("DataControlWidth", New Unit(100, UnitType.Pixels))
vDynamicControl.AddWithMode("DataControl-BackColor", "aliceblue")
```

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

Adding a DES.DynamicControl

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

1. Prepare the page for DES controls. See "Preparing a page for DES controls" in the **General Features Guide**. It covers issues like style sheets, AJAX, and localization.
2. Add an DynamicControl control to the page. It must go in a Template area of a DataBound control, such as ItemTemplate on ListView, a TemplateField object on GridView, or within a Pattern Template file.

Visual Studio and Visual Web Developer Design Mode Users

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

Text Entry Users

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

```
<desDD:DynamicControl id="[YourControlID]" runat="server" />
```

Programmatically creating the DynamicControl control

- Identify the control which you will add the DynamicControl control to its **Controls** collection. Like all ASP.NET controls, the DynamicControl can be added to any control that supports child controls, including Templates.
- Create an instance of the DynamicControl control class. The constructor takes no parameters.
- Assign the **ID** property.
- Add the DynamicControl control to the **Controls** collection.

In this example, the DynamicControl is created with an **ID** of "DynamicControl1". It is added to PlaceHolder1.

[C#]

```
PeterBlum.DES.DynamicData.DynamicControl vDynamicControl =
   new PeterBlum.DES.DynamicData.DynamicControl();
vDynamicControl.ID = "DynamicControl1";
PlaceHolder1.Controls.Add(vDynamicControl);
```

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

[VB]

```
Dim vDynamicControl As PeterBlum.DES.DynamicData.DynamicControl = _
   New PeterBlum.DES.DynamicData.DynamicControl()
vDynamicControl.ID = "DynamicControl1"
PlaceHolder1.Controls.Add(vDynamicControl)
```

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

CONTINUED ON THE NEXT PAGE

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

3. Set the **DataField** property, unless this control is contained in a Pattern Template or Entity Template file.
4. Set the **Mode** property to Edit or Insert, if this control is used in one of those modes.
5. For other properties, see:
   - Selecting the Field Template
   - Supply alternative values for properties on the Field Template

Guidelines for setting properties

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

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

Properties on DES.DynamicControl

Click on any of these topics to jump to them:

- Data Properties
- Field Template Properties
- Data Formatting Properties
- Other Properties

Data Properties

- **DataField** (string) – The data field name for which a Field Template will be generated. This is a required field. However, do not set it when your DynamicControl is inside a Pattern Template or Entity Template because those templates automatically assign it for you. For more, see "Assigning the DataField".

Field Template Properties

- **Mode** (enum System.Web.UI.WebControls.DataBoundControlMode) – Identifies the data entry mode. It is used to select the Field Template from its three forms: *Field Template*.ascx, *Field Template*_Edit.ascx, and *Field Template*_Insert.ascx. It defaults to `DataBoundControlMode.ReadOnly`.
- **UIHint** (string) – The Field Template name. Do not include the file extension (.ascx) or the mode parts: "_Edit" and "_Insert". When set, this value overrides the same property in the UIHintAttribute.
- **DefaultPropertyList** (string) – A property list that is applied the Field Template regardless of the mode. The mode-specific PropertyLists can override identical properties found here. Any unique properties declared here will still be set. This allows you to setup properties common to all modes here.

  For usage, syntax, and programmatic assignment, see "Supply alternative values for properties on the Field Template".
- **ReadOnlyPropertyList** (string) – A property list that is applied to a Field Template in ReadOnly mode. See below for the syntax. This property list is applied after **DefaultPropertyList**, allowing individual properties to override values already applied.

  For usage, syntax, and programmatic assignment, see "Supply alternative values for properties on the Field Template".
- **EditPropertyList** (string) – A property list that is applied to a Field Template in Edit mode. See below for the syntax. This property list is applied after **DefaultPropertyList**, allowing individual properties to override values already applied.

  For usage, syntax, and programmatic assignment, see "Supply alternative values for properties on the Field Template".
- **InsertPropertyList** (string) – A property list that is applied to a Field Template in Insert mode. See below for the syntax. This property list is applied after **DefaultPropertyList**, allowing individual properties to override values already applied.

  For usage, syntax, and programmatic assignment, see "Supply alternative values for properties on the Field Template".
- **DefaultProperties** (PeterBlum.DES.DynamicData.PropertyNamesAndValues) – Holds the list of properties added programmatically with the `AddAsDefault()` method. See "When creating DES.DynamicControl programmatically".

  If the same named property is defined in the mode-specific property below, the mode specific property overrides the value in **DefaultProperties**.
- **ReadOnlyProperties** (PeterBlum.DES.DynamicData.PropertyNamesAndValues) – Holds the list of properties added programmatically with the `AddWithMode()` method by specifying a mode of ReadOnly. See "When creating DES.DynamicControl programmatically".
- **EditProperties** (PeterBlum.DES.DynamicData.PropertyNamesAndValues) – Holds the list of properties added programmatically with the `AddWithMode()` method by specifying a mode of Edit. See "When creating DES.DynamicControl programmatically".

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

- **InsertProperties** (PeterBlum.DES.DynamicData.PropertyNamesAndValues) – Holds the list of properties added programmatically with the `AddWithMode()` method by specifying a mode of Insert. See "When creating DES.DynamicControl programmatically".

Data Formatting Properties

For more details on these properties, see http://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.displayformatattribute_members.aspx.

- **DataFormatString** (string) – A formatting string that specifies the display format of items in the column.
- **NullDisplayText** (string) – The caption displayed for a field when the field's value is null.
- **ConvertEmptyStringToNull** (boolean) – Determines whether empty string values ("") are automatically converted to null values when the data field is updated in the data source.
- **ApplyFormatInEditMode** (boolean) – Determines whether the formatting string specified by the **DataFormatString** property is applied to field values when the data-bound control is in edit mode.
- **HtmlEncode** (boolean) – *Requires Dynamic Data v4 or higher*. Indicates if the text should be Html Encoded before inserting into the page. This will replace various symbols with HTML characters so they appear as symbols in the page. It defaults to `true`.

Other Properties

- **ValidationGroup** (string) – The Validation group name to assign to the validators contained in the Field Template. *ALERT: Do not use this property. It is automatically setup by DES Dynamic Data and is only here because it's defined in the native DynamicControl.*
- **CssClass** (string) – The style sheet class name to enclose the Field Template. It creates a `<span>` tag with this class around the entire Field Template when defined.

  This property is not the best way to setup style sheets. After all, the Field Template contains its own HTML and controls. Set styles there. If you are using Pattern Templates, use NamedStyles to override the default style sheet classes. Otherwise, create public properties in your Field Template that accept the style sheet classes and assign them through the **DefaultPropertyList** property.
- **PatternRequiresDataField** (Boolean) – Used on Pattern Templates to identify if this control gets requires assignment to its **DataField** property by a DataFieldInPattern object.

  If no DataFieldInPattern assigns the **DataField**, when `true`, an exception is thrown. When `false`, the control is made **Visible**=`false`.

  It defaults to `false`.

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

DES.DynamicField: Using Dynamic Data in GridView and DetailsView

The DynamicControl described in the previous section is a placeholder for where a Field Template will generate its content. The DynamicField is a class that makes it easy to setup a DynamicControl when using the GridView and DetailsView controls. It is very similar to the BoundField class, in that it generates HTML content for a specific data field. Since it uses Field Templates, it's a far more flexible way to generate HTML.

As a general rule, substitute DynamicField objects for BoundField objects.

To work, the DynamicField must know the data field name, which you set in its **DataField** property. From there, it can select the correct Field Template and set it up. You can override the default choice for the Field Template (which is based on attributes on the data field's metadata) with the **UIHint** property.

There are actually two DynamicField classes available when using DES Dynamic Data.
`System.Web.DynamicData.DynamicField` is native to ASP.NET Dynamic Data.
`PeterBlum.DES.DynamicData.DynamicField` is a subclass of the native control introduced in DES Dynamic Data. It provides additional features and is the recommended control class. This section describes DES' DynamicField.

Click on any of these topics to jump to them:

- Features of DES.DynamicField
- Using DES.DynamicField
- Properties on DES.DynamicField

Features of DES.DynamicField

- It has separate **UIHint** property values for different data entry modes: **ReadOnlyUIHint**, **EditUIHint**, or **InsertUIHint**.
- Supply alternative values for properties on the Field Template to customize its behavior, making Field Templates very flexible.
- Using your business rules or the value of a UIHint, it loads a Field Template file and generates its content
- Automatic scaffolding generates DynamicFields for GridView and DetailsView controls. (It generates DataFieldInPattern objects for all other situations.)

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

Using DES.DynamicField

The DynamicField has several tasks:

- Identify the data field whose data will be viewed or edited. See "Assigning the DataField"
- Select a Field Template, using your business rules or the value of a UIHint. See "Selecting the Field Template".
- Supply alternative values for properties on the Field Template to customize its behavior. See "Supply alternative values for properties on the Field Template".

Click on any of these topics to jump to them:

- Assigning the DataField
- Selecting the Field Template
- Supply alternative values for properties on the Field Template

Assigning the DataField

The **DataField** property must be assigned in order for DynamicField to do its job.

Use the name of the property, not any name specified on a DisplayNameAttribute assigned to that column.

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="false"
      DataSourceID="GridDataSource" AllowPaging="true"  >
  <Columns>
      <desDD:DynamicField DataField="ColumnName" />
```

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

Selecting the Field Template

Dynamic Data knows exactly which Field Template to use for your data field because you have established that information in the business logic, with the field's actual data type, with the DataTypeAttribute or with the UIHintAttribute.

Sometimes you want to override that choice on the DynamicField. Use the **UIHint** property. Specify the name of the Field Template, omitting the extensions ".ascx", "_edit.ascx", and "_insert.ascx".

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="false"
      DataSourceID="GridDataSource" AllowPaging="true"  >
  <Columns>
      <desDD:DynamicField DataField="ColumnName" UIHint="TemplateName" />
```

The DynamicField may be used for different data entry modes. If you need to select a different Field Template based on the mode, use the **ReadOnlyUIHint**, **EditUIHint**, and **InsertUIHint** properties to specify the desired Field Template.

```
<asp:GridView ID="GridView1" runat="server" AutoGenerateColumns="false"
      DataSourceID="GridDataSource" AllowPaging="true"  >
  <Columns>
      <desDD:DynamicField DataField="ColumnName" EditUIHint="TemplateName" />
```

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

Supply alternative values for properties on the Field Template

ALERT: This feature is only supported by the DES.DynamicField, not the native one.

Each Field Template is derived from a class, and that class offers public properties to customize its behavior. For example, the base class used by all DES Field Templates, `PeterBlum.DES.DynamicData.DESFieldTemplateUserControl`, has these public properties: **DataControlWidth**, **DataControlHeight**, **UseDescriptionAsToolTip**, **UseDependencyAttribute**, and **ValueWhenBlank**. (See "Field Template classes and properties".)

The DES.DynamicField has several properties where you can define a property list for the Field Templates. They are:

- **DefaultPropertyList** – A property list that is applied the Field Template regardless of the mode. See below for the syntax. The mode-specific PropertyLists can override identical properties found here. Any unique properties declared here will still be set. This allows you to setup properties common to all modes here.
- **ReadOnlyPropertyList** – A property list that is applied to a Field Template in ReadOnly mode. See below for the syntax. This property list is applied after **DefaultPropertyList**, allowing individual properties to override values already applied.
- **EditPropertyList** – A property list that is applied to a Field Template in Edit mode. See below for the syntax. This property list is applied after **DefaultPropertyList**, allowing individual properties to override values already applied.
- **InsertPropertyList** – A property list that is applied to a Field Template in Insert mode. See below for the syntax. This property list is applied after **DefaultPropertyList**, allowing individual properties to override values already applied.

Syntax of these properties

The PropertyList properties use the ASP.NET Declarative Syntax, which means it looks like properties you would type into an `<asp:class here>` tag. However, you must enclose every value in single quotes.

Here's the overall pattern:

```
DefaultPropertyList="PropertyName1='propertyvalue' PropertyName2='propertyvalue'"
```

Hint: If you need single quotes within your property values, enter them as "&#39".

```
<desDD:DynamicField DataField="ColumnName"
       InsertPropertyList="DataControlWidth='40px' ValueWhenBlank='FillMe'" />
```

The property value is a string that must be compatible with the TypeConverter for the property's datatype. For example, a DateTime property uses the DateTimeTypeConverter. It converts strings using the ShortDatePattern in the current thread's CultureInfo object or using the culture neutral format "yyyy-MM-dd".

You can refer to the child properties of properties using the syntax: `PropertyName-ChildPropertyName='value'`. This is important because you have access to the data control through the **DataControl** property and the ColumnValidatorManager through the **ColumnValidatorManager** property.

For example, `InsertPropertyList="DataControl-BackColor='aliceblue'"`.

When you declare an unknown property name, it will be ignored. If you want to see any errors caused by incorrect property names or values, set `<% @Page Trace="true" %>` and view the trace.

Please refer to "Field Template classes and properties" to learn about the available properties.

Examples

```
<desDD:DynamicField DataField="ColumnName"
       DefaultPropertyList="DataControlWidth='100px'"
       InsertPropertyList="DataControlWidth='80px' ValueWhenBlank='FillMe'" />

<desDD:DynamicField DataField="ColumnName"
       DefaultPropertyList="UseDescriptionAsToolTip='false'" />

<desDD:DynamicField DataField="ColumnName"
       InsertPropertyList="DataControl-BackColor='aliceblue'" />
```

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

When creating DES.DynamicField programmatically

When you create or modify a DynamicField object programmatically, don't assign values to the above properties because those properties require parsing and conversion to their native types. That uses extra CPU time that can be avoided by using the following technique.

The DynamicField class defines these properties to be used programmatically: **DefaultProperties**, **ReadOnlyProperties**, **EditProperties**, and **InsertProperties**. Each of these are a collection which take a property name (as a case sensitive string) and a value (in the actual type expected by the property, like an integer or boolean).

Call the method `AddAsDefault()` to add to the **DefaultProperties** collection. Call the method `AddWithMode()` to add to any of the other properties.

[C#]

```
void AddAsDefault(string pPropertyName, object pValue)
```

```
void AddWithMode(System.Web.UI.WebControls.DataBoundControlMode pMode,
     string pPropertyName, object pValue)
```

[VB]

```
Sub AddAsDefault(ByVal pPropertyName As String, ByVal pValue As Object)
```

```
Sub AddWithMode(ByVal pMode As System.Web.UI.WebControls.DataBoundControlMode, _
     ByVal pPropertyName As String, ByVal pValue As Object)
```

Parameters

pPropertyName

The name of the property on the Field Template. It must be a case sensitive match.

pValue

The value to pass to that property in the actual type of the property.

pMode

Selects one of these properties to hold the property value: **ReadOnlyProperties**, **EditProperties**, and **InsertProperties**.

Example

[C#]

```
using DES=PeterBlum.DES.DynamicData;
…
DES.DynamicField vDynamicField = new DES.DynamicField();
vDynamicField.AddAsDefault("DataControlWidth", new Unit(100, UnitType.Pixels));
vDynamicField.AddWithMode("DataControl-BackColor", "aliceblue");
```

[VB]

```
Imports DES=PeterBlum.DES.DynamicData
…
Dim vDynamicField As DES.DynamicField = New DES.DynamicField()
vDynamicField.AddAsDefault("DataControlWidth", New Unit(100, UnitType.Pixels))
vDynamicField.AddWithMode("DataControl-BackColor", "aliceblue")
```

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

Properties on DES.DynamicField

Click on any of these topics to jump to them:

- Data Properties
- Data Formatting Properties
- Field Template Properties
- Appearance Properties

Data Properties

- **DataField** (string) – The data field name for which a Field Template will be generated. This is a required field. However, do not set it when your DynamicField is inside a Pattern Template or Entity Template because those templates automatically assign it for you. For more, see "Assigning the DataField".
- **ReadOnly** (boolean) – Force the Field Template to be in a ReadOnly mode even if others are in Edit or Insert modes.

  When `true`, the field is readonly.

  It defaults to `false`.

Data Formatting Properties

For more details on these properties, see http://msdn.microsoft.com/en-us/library/system.componentmodel.dataannotations.displayformatattribute_members.aspx.

- **DataFormatString** (string) – A formatting string that specifies the display format of items in the column.
- **NullDisplayText** (string) – The caption displayed for a field when the field's value is null.
- **ConvertEmptyStringToNull** (boolean) – Determines whether empty string values ("") are automatically converted to null values when the data field is updated in the data source.
- **ApplyFormatInEditMode** (boolean) – Determines whether the formatting string specified by the **DataFormatString** property is applied to field values when the data-bound control is in edit mode.
- **HtmlEncode** (boolean) – *Requires Dynamic Data v4 or higher*. Indicates if the text should be Html Encoded before inserting into the page. This will replace various symbols with HTML characters so they appear as symbols in the page. It defaults to `true`.

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

Field Template Properties

- **UIHint** (string) – The Field Template name. Do not include the file extension (.ascx) or the mode parts: "_Edit" and "_Insert". When set, this value overrides the same property in the UIHintAttribute.
- **ReadOnlyUIHint** (string) – The Field Template name requested for ReadOnly mode. If blank, use the **UIHint** property.
- **EditUIHint** (string) – The Field Template name requested for Edit mode. If blank, use the **UIHint** property.
- **InsertUIHint** (string) – The Field Template name requested for Insert mode. If blank, use the **UIHint** property.
- **DefaultPropertyList** (string) – A property list that is applied the Field Template regardless of the mode. The mode-specific PropertyLists can override identical properties found here. Any unique properties declared here will still be set. This allows you to setup properties common to all modes here.

  For usage, syntax, and programmatic assignment, see "Supply alternative values for properties on the Field Template".
- **ReadOnlyPropertyList** (string) – A property list that is applied to a Field Template in ReadOnly mode. See below for the syntax. This property list is applied after **DefaultPropertyList**, allowing individual properties to override values already applied.

  For usage, syntax, and programmatic assignment, see "Supply alternative values for properties on the Field Template".
- **EditPropertyList** (string) – A property list that is applied to a Field Template in Edit mode. See below for the syntax. This property list is applied after **DefaultPropertyList**, allowing individual properties to override values already applied.

  For usage, syntax, and programmatic assignment, see "Supply alternative values for properties on the Field Template".
- **InsertPropertyList** (string) – A property list that is applied to a Field Template in Insert mode. See below for the syntax. This property list is applied after **DefaultPropertyList**, allowing individual properties to override values already applied.

  For usage, syntax, and programmatic assignment, see "Supply alternative values for properties on the Field Template".
- **DefaultProperties** (PeterBlum.DES.DynamicData.PropertyNamesAndValues) – Holds the list of properties added programmatically with the `AddAsDefault()` method. See "When creating DES.DynamicField programmatically".

  If the same named property is defined in the mode-specific property below, the mode specific property overrides the value in **DefaultProperties**.
- **ReadOnlyProperties** (PeterBlum.DES.DynamicData.PropertyNamesAndValues) – Holds the list of properties added programmatically with the `AddWithMode()` method by specifying a mode of ReadOnly.

  See "When creating DES.DynamicField programmatically".
- **EditProperties** (PeterBlum.DES.DynamicData.PropertyNamesAndValues) – Holds the list of properties added programmatically with the `AddWithMode()` method by specifying a mode of Edit.

  See "When creating DES.DynamicField programmatically".
- **InsertProperties** (PeterBlum.DES.DynamicData.PropertyNamesAndValues) – Holds the list of properties added programmatically with the `AddWithMode()` method by specifying a mode of Insert.

  See "When creating DES.DynamicField programmatically".

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

Appearance Properties

These properties are from the base class. See http://msdn.microsoft.com/en-us/library/system.web.dynamicdata.dynamicfield_properties.aspx.

- **AccessibleHeaderText** (string) – Gets or sets text that is rendered as the AbbreviatedText property value in some controls. (Inherited from DataControlField.)
- **HeaderText** (string) – Gets or sets the text that is displayed in the header of the data-bound control that contains the DynamicField object.
- **FooterText** (string) – Gets or sets the text that is displayed in the footer item of a data control field.
- **HeaderStyle** (Style) – Gets or sets the style of the header of the data control field.
- **FooterStyle** (Style) – Gets or sets the style of the footer of the data control field.
- **ShowHeader** (Boolean) – Gets or sets a value indicating whether the header item of a data control field is rendered.
- **HeaderImageUrl** (String) – Gets or sets the URL of an image that is displayed in the header item of a data control field.
- **ItemStyle** (Style) – The style sheet class name to enclose the Field Template. It creates a `<span>` tag with this class around the entire Field Template when defined.

  This property is not the best way to setup style sheets. After all, the Field Template contains its own HTML and controls. Set styles there. If you want to control these values on the DynamicField object, create public properties in your Field Template that accept the style sheet classes and assign them through the **DefaultPropertyList** property.

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

Behavior Properties

These properties are from the base class. See http://msdn.microsoft.com/en-us/library/system.web.dynamicdata.dynamicfield_properties.aspx.

- **SortExpression** (string) – Gets or sets the sort expression that is used when the data field is used to sort the data source by.
- **InsertVisible** (Boolean) – Gets a value indicating whether the DataControlField object is visible when its parent data-bound control is in insert mode.

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

Customizer Control

The Customizer control customizes the behavior of DES-supplied Field Templates and Page Templates. Use it to avoid editing the actual Field- and Page Templates in many cases. For example, if you want the Currency_Edit.ascx Field Template to show the currency symbol in its CurrencyTextBox, the Customizer offers the property **CurrencyTextBox_UseCurrencySymbol**. The Customizer's **ErrorMessages** property is one of its most valuable as it is where you can define the error messages for your validators, again without editing the actual Field Templates.

For an example using the Customizer on a master page for your Page Templates:

[C#]

[DES Product Folder]/DynamicData/CS/Samples/Site.master

[VB]

[DES Product Folder]/DynamicData/VB/Samples/Site.master

Click on any of these topics to jump to them:

- Using the Customizer Control
- Properties on the Customizer Control

Example

```
<desDD:Customizer ID="Customizer1" runat="server"
   CurrencyTextBox_AlignRight="True" CurrencyTextBox_ShowSpinner="True"
   DecimalTextBox_AlignRight="True" DecimalTextBox_ShowSpinner="True"
   IntegerTextBox_ShowSpinner="True" DisableOnSubmit="True">
   <ErrorMessages>
      <desDD:ValidatorErrorMessages NameOfValidatorType="RequiredTextValidator"
         ErrorMessage="Required" />
      <desDD:ValidatorErrorMessages NameOfValidatorType="DataTypeCheckValidator"
         DataTypeName="Date" ErrorMessage="Date format is yyyy-MM-dd" />
      <desDD:ValidatorErrorMessages NameOfValidatorType="DataTypeCheckValidator"
         DataTypeName="Integer" ErrorMessage="Enter only digits" />
   </ErrorMessages>
</desDD:Customizer>
```

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

Using the Customizer Control

Add the Customizer control and set the desired properties. See "Properties on the Customizer Control".

You can have multiple Customizer controls. Each DataBound Control Adapter (declared in the DynamicDataManager control) can have a separate one. All controls contained within the DataBound control, such as Field Templates, Pattern Templates, and Buttons, share a common Customizer. Controls within different DataBound controls can have different Customizers.

Specifying the Customizer on the DataBound Control Adapter

The DataBound Control Adapter will locate the appropriate Customizer control if you do not explicitly assign it to the Adapter's **CustomizerControlID** property. Here are some recommendations:

- Assign the **CustomizerControlID** property when you want to explicity control which Customizer is used. DES searches for a control with that ID in the DynamicDataManager's current NamingContainer. If not found there, it looks up through the ancestor NamingContainers. So your DynamicDataManager can be in a UserControl and it will find the Customizer in the Page definition.

```
<desDD:Customizer ID="Customizer1" runat="server"
   CurrencyTextBox_AlignRight="True" CurrencyTextBox_ShowSpinner="True"
   DecimalTextBox_AlignRight="True" DecimalTextBox_ShowSpinner="True"
   IntegerTextBox_ShowSpinner="True" DisableOnSubmit=True>
</desDD:Customizer>

<desDD:DynamicDataManager ID="DynamicDataManager1" runat="server"
   AutoLoadForeignKeys="True">
   <Adapters>
      <desDD:FormViewAdapter DataBoundControlID="FormView1"
         ValidationSummaryControlID="ValidationSummary1"
         SupportsEditMode="true" SupportsInsertMode="true"
         CustomizerControlID="Customizer1"  />
   </Adapters>
</desDD:DynamicDataManager>
```

- When programmatically defining a DataBound Control Adapter, assign a reference to the **CustomizerControl** property on the DataBound Control Adapter instead of using the **CustomizerControlID** property for better performance.

  [C#]

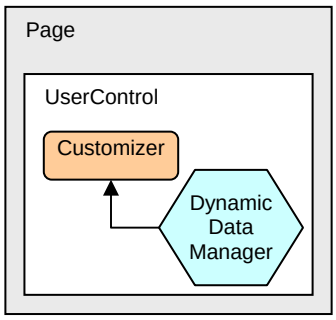
```
DynamicDataManager1.Adapters[0].CustomizerControl = Customizer1;
```

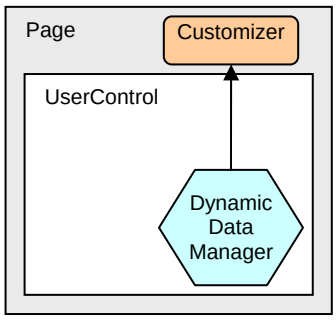
  [VB]

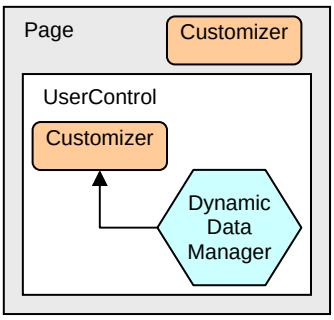
```
DynamicDataManager1.Adapters(0).CustomizerControl = Customizer1
```

- When neither **CustomizerControlID** or **CustomizeControl** properties are assigned, the DynamicDataManager searches up the control tree until it finds a Customizer or hits the Page node. So if you have a Customizer on the Page node and also on the same UserControl as the DynamicDataManager, the DynamicDataManager will find the one on the same UserControl.

  Here are examples for searching when the DynamicDataManager is in a UserControl.







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

Properties on the Customizer Control

Click on any of these topics to jump to them:

- Date and Time Field Template Properties
- Numeric Field Template Properties
- Text Field Template Properties
- Button Properties
- Validation Properties
- ForeignKey Field Template Properties
- Miscellaneous Properties

Date and Time Field Template Properties

- **DateTextBox_DateFormat** (System.Nullable<PeterBlum.DES.EntryDateFormat>) – Determines the date format used by DateTextBox.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.DateEditFTUC`, which includes **Date_Edit.ascx** and **DateTime_Edit.ascx**.

  There are 3 sources for the DateFormat:

  - First check the **DateFormat** property on the Field Template. If its not `null`, use that.
  - Check **DateTextBox_DateFormat** (here). If its not `null`, use it.
  - Use the value on the **DateTextBox.DateFormat** itself.

  Since both **DateEditFTUC.DateFormat** and **DateTextBox_DateFormat** default to `null`, the **DateTextBox.DateFormat** is not modified by default.

  It defaults to `null` (use the DateFormat on DateTextBox).

- **DateTextBox_AllowMonthNames** (System.Nullable<PeterBlum.DES.AllowMonthNames>) – Determines if the short date format shows digits or the abbreviated month name for the month.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.DateEditFTUC`, which includes **Date_Edit.ascx** and **DateTime_Edit.ascx**.

- **MonthYearTextBox_DateFormat** (System.Nullable<PeterBlum.DES.EntryDateFormat>) – Determines the date format used by MonthYearTextBox.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.MonthYearEditFTUC`, which includes **MonthYear_Edit.ascx**.

  There are 3 sources for the DateFormat:

  - First check **DateFormat** property on the Field Template. If its not `null`, use that.
  - Check **MonthYearTextBox_DateFormat** (here). If its not `null`, use it.
  - Use the value on the **MonthYearTextBox.DateFormat** itself.

  Since both **MonthYearEditFTUC.DateFormat** and **MonthYearTextBox_DateFormat** default to `null`, the **MonthYearTextBox.DateFormat** is not modified by default.

  It defaults to `null` (use the DateFormat on MonthYearTextBox).

- **MonthYearTextBox_AllowMonthNames** (System.Nullable<PeterBlum.DES.AllowMonthNames>) – Determines if the short date format shows digits or the abbreviated month name for the month.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.MonthYearEditFTUC`, which includes **MonthYear_Edit.ascx**.

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

- **AnniversaryTextBox_DateFormat** (System.Nullable<PeterBlum.DES.EntryDateFormat>) – Determines the date format used by AnniversaryTextBox.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.AnniversaryEditFTUC`, which includes **Anniversary_Edit.ascx**.

  There are 3 sources for the DateFormat:

  - First check **DateFormat** property on the Field Template. If its not `null`, use that.
  - Check **AnniversaryTextBox_DateFormat** (here). If its not `null`, use it.
  - Use the value on the **AnniversaryTextBox.DateFormat** itself.

  Since both **AnniversaryEditFTUC.DateFormat** and **AnniversaryTextBox_DateFormat** default to `null`, the **AnniversaryTextBox.DateFormat** is not modified by default.

  It defaults to `null` (use the DateFormat on AnniversaryTextBox).

- **AnniversaryTextBox_AllowMonthNames** (System.Nullable<PeterBlum.DES.AllowMonthNames>) – Determines if the short date format shows digits or the abbreviated month name for the month.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.AnniversaryEditFTUC`, which includes **Anniversary_Edit.ascx**.

- **Label_DateFormat** (System.Nullable<PeterBlum.DES.EntryDateFormat>) – Determines the date format used by a Label.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.BaseDateStaticFTUC`, which includes **Date.ascx**, **DateTime.ascx**, **Anniversary.ascx**, and **MonthYear.ascx**.

  There are 2 sources for the DateFormat:

  - First check **DateFormat** property on the Field Template. If its not `null`, use that.
  - Check **DateTextBox_DateFormat** (here). If its not `null`, use it.

  Since both **DateEditFTUC.DateFormat** and **DateTextBox_DateFormat** default to `null`, the **DateTextBox.DateFormat** is not modified by default.

  It defaults to `null` (use the DateFormat on DateTextBox).

- **Label_AllowMonthNames** (System.Nullable<PeterBlum.DES.AllowMonthNames>) – Determines if the short date format shows digits or the abbreviated month name for the month.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.BaseDateStaticFTUC`, which includes **Date.ascx**, **DateTime.ascx**, **Anniversary.ascx**, and **MonthYear.ascx**.

- **TimeTextBox_ShowSeconds** (System.Nullable<PeterBlum.DES.BaseTimeTextBox.ShowSecondsMode>) – Determines if a TimeOfDayTextBox or DurationTextBox show seconds.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.BaseDateTimeEditFTUC`, which includes **Time_Edit.ascx** , **DateTime_Edit.ascx**, and **Duration_Edit.ascx**.

- **Label_ShowSeconds** (System.Nullable<PeterBlum.DES.BaseTimeTextBox.ShowSecondsMode>) – Determines if a Label showing time or duration shows seconds.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.BaseDateTimeFTUC`, which includes **Time.ascx** , **DateTime.ascx**, and **Duration.ascx**.

- **TimeTextBox_ShowSpinners** (System.Nullable<PeterBlum.DES.BaseTimeTextBox.ShowCommandButtons>) – Determines if a TimeOfDayTextBox or DurationTextBox show the spinner controls that adjust the hours, minutes and seconds.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.BaseDateTimeEditFTUC`, which includes **Time_Edit.ascx** , **DateTime_Edit.ascx**, and **Duration_Edit.ascx**.

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

- **Date_ReadOnlyAllowsEdits** (bool) – When `true`, it sets up the DateTextBox, MonthYearTextBox, and AnniversaryTextBox as **ReadOnly** but sets the **ReadOnlyAllowsEdits** property to support the popup calendar, context menu, and keyboard commands as editors.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.BaseDateStaticFTUC`, which includes **Date.ascx**, **DateTime.ascx**, **Anniversary.ascx**, and **MonthYear.ascx**.

  When `true`, the mode is used. When `false`, the setting on the textboxes is unchanged.

- **Time_ReadOnlyAllowsEdits** (bool) – When `true`, it sets up the TimeOfDayTextBox and DurationTextBox as **ReadOnly** but sets the **ReadOnlyAllowsEdits** property to support the spinners, context menu, and keyboard commands as editors.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.BaseDateTimeFTUC`, which includes **Time.ascx** , **DateTime.ascx**, and **Duration.ascx**.

  When `true`, the mode is used. When `false`, the setting on the textboxes is unchanged.

- **DateTime_HintPopupViewName** (string) – Determines how the auto hint feature appears on Date and Time textboxes. Specify the name of a Hint PopupView or use "{DEFAULT}" to use the page default. If blank, popup views are NOT used. Instead, tooltips are used.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.BaseDateTimeEditFTUC`, which includes **Date_Edit.ascx**, **MonthYear_Edit.ascx**, **Anniversary_Edit.ascx**, **Time_Edit.ascx** , **DateTime_Edit.ascx**, and **Duration_Edit.ascx**.

  It defaults to `"{DEFAULT}"`.

- **DateTime_AutoHintStyle** (System.Nullable<PeterBlum.DES.AutoHintStyle>) – Determines the setting for the AutoHintStyle property on any date and time textbox. When `null`, the control's setting is not changed from what was assigned on the Field Template.

  The enumerated type `PeterBlum.DES.AutoHintStyle` has these values:

  - `Default` - Use the **DefaultAutoHintStyle** property in the "HintManager Settings" topic of the **Global Settings Editor**. **DefaultAutoHintStyle** is the same enumerated type, `PeterBlum.DES.AutoHintStyle`, and defaults to `HintFormatter`. This is the default. As a result, it is easy to customize all of your Date and Time textboxes quickly, by changing one setting in the **Global Settings Editor**.

    *Note: If selected in the **DefaultAutoHintStyle** property, it uses HintFormatter.*

  - `None` - Turns off the Automatic Hint feature. Effectively it is the same as setting **AutoHint** to `false`.

  - `ToolTip` - Uses the tooltip only. The tooltip is usually a browser specific popup. However, when **HintManager.EnableToolTipsAsPopupViews** is `true`, it uses the Enhanced Tooltip.

  - `HintFormatter` - Follows the rules of the **LocalHintFormatter** or **SharedHintFormatterName** properties of the textbox.

  - `PopupViewHint` - Sets up a PopupView for the hint, with no tooltip or status bar to show the same text. The PopupView name is determined by the **DefaultHintPopupViewName** property in the "HintManager Settings" topic of the **Global Settings Editor**.

  - `PopupViewHintAndToolTip` - Sets up a PopupView for the hint and tooltip. The status bar is not used to show the same text. The PopupView name is determined by the **DefaultHintPopupViewName** property in the "HintManager Settings" topic of the **Global Settings Editor**.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.BaseDateTimeEditFTUC`, which includes **Date_Edit.ascx**, **MonthYear_Edit.ascx**, **Anniversary_Edit.ascx**, **Time_Edit.ascx** , **DateTime_Edit.ascx**, and **Duration_Edit.ascx**.

  It defaults to `null`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Numeric Field Template Properties

- **Label_ShowThousandsSeparator** (enum PeterBlum.DES.TrueFalseIgnore) – Show the thousands separator in Labels that display numbers. Only applies to the default format. If the user uses the **DataFormatString** property on the DisplayFormatAttribute, it takes precedence.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.BaseNumericFTUC`, which includes **Integer.ascx** , **Decimal.ascx**, **Currency.ascx**, **Percent.ascx**, and **Measurement.ascx**.

  It defaults to `TrueFalseIgnore.Ignore`.

- **IntegerTextBox_ShowThousandsSeparator** (enum PeterBlum.DES.TrueFalseIgnore) – Show the thousands separator in IntegerTextBoxes.

  Supported by any Field Template subclassed from PeterBlum.DES.DynamicData.IntegerEditFTUC, which includes **Integer_Edit.ascx** and from `PeterBlum.DES.DynamicData.MeasurementEditFTUC` which includes **Measurement_Edit.ascx**.

  It defaults to `TrueFalseIgnore.Ignore`.

- **DecimalTextBox_ShowThousandsSeparator** (enum PeterBlum.DES.TrueFalseIgnore) – Show the thousands separator in DecimalTextBoxes.

  Supported by any Field Template subclassed from PeterBlum.DES.DynamicData.DecimalEditFTUC, which includes **Decimal_Edit.ascx** and from `PeterBlum.DES.DynamicData.MeasurementEditFTUC` which includes **Measurement_Edit.ascx**.

  It defaults to `TrueFalseIgnore.Ignore`.

- **CurrencyTextBox_ShowThousandsSeparator** (enum PeterBlum.DES.TrueFalseIgnore) – Show the thousands separator in CurrencyTextBoxes.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.CurrencyEditFTUC`, which includes **Currency_Edit.ascx**.

  It defaults to `TrueFalseIgnore.Ignore`.

- **PercentTextBox_ShowThousandsSeparator** (enum PeterBlum.DES.TrueFalseIgnore) – Show the thousands separator in PercentTextBoxes.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.PercentEditFTUC`, which includes **Percent_Edit.ascx**.

  It defaults to `TrueFalseIgnore.Ignore`.

- **IntegerTextBox_ShowSpinner** (enum PeterBlum.DES.TrueFalseIgnore) – Show the spinner control on IntegerTextBoxes.

  Supported by any Field Template subclassed from PeterBlum.DES.DynamicData.IntegerEditFTUC, which includes **Integer_Edit.ascx** and from `PeterBlum.DES.DynamicData.MeasurementEditFTUC` which includes **Measurement_Edit.ascx**.

  It defaults to `TrueFalseIgnore.Ignore`.

- **DecimalTextBox_ShowSpinner** (enum PeterBlum.DES.TrueFalseIgnore) – Show the spinner control on DecimalTextBoxes.

  Supported by any Field Template subclassed from PeterBlum.DES.DynamicData.DecimalEditFTUC, which includes **Decimal_Edit.ascx** and from `PeterBlum.DES.DynamicData.MeasurementEditFTUC` which includes **Measurement_Edit.ascx**.

  It defaults to `TrueFalseIgnore.Ignore`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **CurrencyTextBox_ShowSpinner** (enum PeterBlum.DES.TrueFalseIgnore) – Show the spinner control on CurrencyTextBoxes.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.CurrencyEditFTUC`, which includes **Currency_Edit.ascx**.

  It defaults to `TrueFalseIgnore.Ignore`.

- **PercentTextBox_ShowSpinner** (enum PeterBlum.DES.TrueFalseIgnore) – Show the spinner control on PercentTextBoxes.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.PercentEditFTUC`, which includes **Percent_Edit.ascx**.

  It defaults to `TrueFalseIgnore.Ignore`.

- **Label_UseCurrencySymbol** (enum PeterBlum.DES.TrueFalseIgnore) – Show the currency symbol in Labels that display numbers. Only applies to the default format. If the user uses the **DataFormatString** property on the DisplayFormatAttribute, it takes precedence.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.BaseCurrencyStaticFTUC`, which includes **Currency.ascx**.

  It defaults to `TrueFalseIgnore.Ignore`.

- **CurrencyTextBox_UseCurrencySymbol** (enum PeterBlum.DES.TrueFalseIgnore) – Show the currency symbol in CurrencyTextBoxes.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.CurrencyEditFTUC`, which includes **Currency_Edit.ascx**.

  It defaults to `TrueFalseIgnore.Ignore`.

- **Label_UsePercentSymbol** (enum PeterBlum.DES.TrueFalseIgnore) – Show the Percent symbol in Labels that display numbers. Only applies to the default format. If the user uses the **DataFormatString** property on the DisplayFormatAttribute, it takes precedence.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.BasePercentStaticFTUC`, which includes **Percent.ascx**.

  It defaults to `TrueFalseIgnore.Ignore`.

- **PercentTextBox_UsePercentSymbol** (enum PeterBlum.DES.TrueFalseIgnore) – Show the Percent symbol in PercentTextBoxes.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.PercentEditFTUC`, which includes **Percent_Edit.ascx**.

  It defaults to `TrueFalseIgnore.Ignore`.

- **IntegerTextBox_AlignRight** (enum PeterBlum.DES.TrueFalseIgnore) – Right align the text in IntegerTextBoxes.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.IntegerEditFTUC`, which includes **Integer_Edit.ascx** and from `PeterBlum.DES.DynamicData.MeasurementEditFTUC` which includes **Measurement_Edit.ascx**.

  It defaults to `TrueFalseIgnore.Ignore`.

- **DecimalTextBox_AlignRight** (enum PeterBlum.DES.TrueFalseIgnore) – Right align the text in DecimalTextBoxes.

  Supported by any Field Template subclassed from PeterBlum.DES.DynamicData.DecimalEditFTUC, which includes **Decimal_Edit.ascx** and from `PeterBlum.DES.DynamicData.MeasurementEditFTUC` which includes **Measurement_Edit.ascx**.

  It defaults to `TrueFalseIgnore.Ignore`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **CurrencyTextBox_AlignRight** (enum PeterBlum.DES.TrueFalseIgnore) – Right align the text in CurrencyTextBoxes.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.CurrencyEditFTUC`, which includes **Currency_Edit.ascx**.

  It defaults to `TrueFalseIgnore.Ignore`.

- **PercentTextBox_AlignRight** (enum PeterBlum.DES.TrueFalseIgnore) – Right align the text in PercentTextBoxes.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.PercentEditFTUC`, which includes **Percent_Edit.ascx**.

  It defaults to `TrueFalseIgnore.Ignore`.

- **IntegerTextBox_SmartColumns** (enum PeterBlum.DES.DynamicData.SmartNumericColumns) – Calculate the TextBox **Columns** property based on a reasonable maximum size for the number on IntegerTextBoxes. See the SmartColumns property for the rules used in the calculation.

  Only used when the Field Template's own **SmartColumns** property is `SmartNumericColumns.Default`.

  The enumerated type `PeterBlum.DES.DynamicData.SmartNumericColumns` has these values:

  - `Never` - Do not modify the existing value of **Columns**.
  - `WhenZero` - Only modify it when its 0. This is the default.
  - `Always` - Modify it.
  - `Default` – Do not use this value.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.IntegerEditFTUC`, which includes **Integer_Edit.ascx** and from `PeterBlum.DES.DynamicData.MeasurementEditFTUC` which includes **Measurement_Edit.ascx**.

- **DecimalTextBox_SmartColumns** (enum PeterBlum.DES.DynamicData.SmartNumericColumns) – Calculate the TextBox **Columns** property based on a reasonable maximum size for the number on DecimalTextBoxes. See the SmartColumns property for the rules used in the calculation.

  Only used when the Field Template's own **SmartColumns** property is `SmartNumericColumns.Default`.

  The enumerated type `PeterBlum.DES.DynamicData.SmartNumericColumns` has these values:

  - `Never` - Do not modify the existing value of **Columns**.
  - `WhenZero` - Only modify it when its 0. This is the default.
  - `Always` - Modify it.
  - `Default` – Do not use this value.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.DecimalEditFTUC`, which includes **Decimal_Edit.ascx** and from `PeterBlum.DES.DynamicData.MeasurementEditFTUC` which includes **Measurement_Edit.ascx**.

  It defaults to `SmartNumericColumns.WhenZero`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **CurrencyTextBox_SmartColumns** (enum PeterBlum.DES.DynamicData.SmartNumericColumns) – Calculate the TextBox **Columns** property based on a reasonable maximum size for the number on CurrencyTextBoxes. See the SmartColumns property for the rules used in the calculation.

  Only used when the Field Template's own **SmartColumns** property is `SmartNumericColumns.Default`.

  The enumerated type `PeterBlum.DES.DynamicData.SmartNumericColumns` has these values:

  - `Never` - Do not modify the existing value of **Columns**.
  - `WhenZero` - Only modify it when its 0. This is the default.
  - `Always` - Modify it.
  - `Default` – Do not use this value.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.CurrencyEditFTUC`, which includes **Currency_Edit.ascx**.

  It defaults to `SmartNumericColumns.WhenZero`.

- **PercentTextBox_SmartColumns** (enum PeterBlum.DES.DynamicData.SmartNumericColumns) – Calculate the TextBox **Columns** property based on a reasonable maximum size for the number on PercentTextBoxes. See the SmartColumns property for the rules used in the calculation.

  Only used when the Field Template's own **SmartColumns** property is `SmartNumericColumns.Default`.

  The enumerated type `PeterBlum.DES.DynamicData.SmartNumericColumns` has these values:

  - `Never` - Do not modify the existing value of **Columns**.
  - `WhenZero` - Only modify it when its 0. This is the default.
  - `Always` - Modify it.
  - `Default` – Do not use this value.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.PercentEditFTUC`, which includes **Percent_Edit.ascx**.

  It defaults to `SmartNumericColumns.WhenZero`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Text Field Template Properties

- **UseTextCounter** (enum PeterBlum.DES.DynamicData.UseTextCounter) – Field Templates that use a TextBox can optionally use the DES TextCounter control to assist the user as they type. This property determines when the TextCounter is enabled.

  The enumerated type `PeterBlum.DES.DynamicData.UseTextCounter` has these values:

  - `Ignore` – Take no action. The properties of the TextCounter control on the Field Template determine its state.
  - `None` – Do not use the TextCounter.
  - `Always` – Use the TextCounter
  - `MaxLength` – Use the TextCounter when the column has a maximum length established. This is the default.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.BaseTextBoxFTUC`, which includes **Text_Edit.ascx**, **Text_Insert.ascx**, **MultilineText_Edit.ascx** and **MultilineText_Insert.ascx**.

- **Text_Columns** (Integer) – Determines the **Columns** property value for singleline TextBoxes that are used with Text datatypes when **Columns** has not been assigned by any other technique. When 0, it is not used. It defaults to 0.

  Its behavior is modified by the **TextColumnsMode** property, below.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.TextEditFTUC`, which includes **Text_Edit.ascx** and**Text_Insert.ascx**.

- **TextColumnsMode** (enum PeterBlum.DES.DynamicData.TextColumnsMode) – When using **Text_Columns**, this determines if its size is exact or dependent on the column's maximum size.

  The enumerated type `PeterBlum.DES.DynamicData.TextColumnsMode` has these values:

  - `Exact` – Always use the value of **Text_Columns**.
  - `Maximum` – Use the length of the column from metadata until it reaches **Text_Columns**. Then use **Text_Columns**. This is the default.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.TextEditFTUC`, which includes **Text_Edit.ascx** and**Text_Insert.ascx**.

- **MultlineText_Columns** (Integer) – Determines the **Columns** property value for multline TextBoxes that are used with Text datatypes when **Columns** has not been assigned by any other technique. When 0, it is not used.

  It defaults to 80.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.MultlineTextEditFTUC`, which includes **MultilineText_Edit.ascx** and **MultilineText_Insert.ascx**.

- **MultlineText_Rows** (Integer) – Determines the **Rows** property value for multline TextBoxes that are used with Text datatypes when **Rows** has not been assigned by any other technique. When 0, it is not used.

  It defaults to 5.

  Supported by any Field Template subclassed from `PeterBlum.DES.DynamicData.MultlineTextEditFTUC`, which includes **MultilineText_Edit.ascx** and **MultilineText_Insert.ascx**.

- **Text_ShowUrlsAsHyperlinks** (PeterBlum.DES.TrueFalseDefault) – Determines if URLs within the text are converted to hyperlinks.

  The enumerated type `PeterBlum.DES.TrueFalseDefault` has these values:

  - `True` – Convert to hyperlinks.
  - `False` – Do not convert.
  - `Default` – Use the business rules on the column's metadata. It will convert to hyperlinks when the DataTypeAttribute is set to `Multiline` or the length is very long (or not defined at all). This is the default.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **Text_ShowEmailsAsHyperlinks** (PeterBlum.DES.TrueFalseDefault) – Determines if email addresses within the text are converted to hyperlinks.

  The enumerated type `PeterBlum.DES.TrueFalseDefault` has these values:

  - `True` – Convert to hyperlinks.
  - `False` – Do not convert.
  - `Default` – Use the business rules on the column's metadata. It will convert to hyperlinks when the DataTypeAttribute is set to `Multiline` or the length is very long (or not defined at all). This is the default.

- **Text_HintPopupViewName** (string) – Determines how the hint feature appears on most textboxes, except Date and Time. Specify the name of a Hint PopupView or use "{DEFAULT}" to use the page default. If blank, popup views are NOT used. Instead, tooltips are used.

  When set to "", it does not preregister the hints feature in the AJAXManager.

  This is a performance optimization.

  It defaults to "`{DEFAULT}`".

- **FormattedString_ShowExampleInHint** (boolean) – Formatted strings may offer an example of their formatting taken from their DataTypeAttributes (like EmailAddressDataTypeAddress and UrlDataTypeAddress). Use this to display that example in a hint or tooltip for the textbox.

  If no DataTypeAttribute is assigned to the column, no example will be applied.

  You can replace the default example by setting the **FormattingExample** property on the DataTypeAttribute.

  When `true`, the example is shown.

  When `false`, it is not.

  It defaults to `true`.

- **FormattedString_ExampleFormatter** (string) – When **FormattedString_ShowExampleInHint** is true, the example text can be enclosed in this string so long as it contains the {EXAMPLE} token where the example text goes.

  If this is blank, the example text is the only text used. If the {EXAMPLE} token is not available, the example text is appended.

  It defaults to "`<em>Example:</em>{NEWLINE}{EXAMPLE}`".

- **EmailAddress_ShowAsHyperlink** (boolean) – Used on the EmailAddress Field Template to display the email address as a hyperlink.

  You can setup the Field Template with an actual Hyperlink control but that cannot show more than one email address when you have setup the field to allow multiple email addresses. Instead, use a Label, LiteralControl, or a PlaceHolder and the Field Template will create a string with the correct HTML for that control.

  Used by Field Templates that subclass from `PeterBlum.DES.DynamicData.EmailAddressStaticFTUC`, including Email.ascx.

  When `true`, use a hyperlink.

  When `false`, do not edit the field's value.

  It defaults to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Button Properties

- **DisableOnSubmit** (enum PeterBlum.DES.TrueFalseIgnore) – Determines if the buttons in DetailViews and GridViews will disable on submit. Other DataBound controls explicitly declare their buttons and can setup individual <des:Button> tags with **DisableOnSubmit** explicitly definied.

  This property has these values:

  - `True` – Set **DisableOnSubmit** to `true`.
  - `False` – Set **DisableOnSubmit** to `false`, overriding any existing setting of true on the control.
  - `Ignore` – Use the existing property setting on the control. This is the default.

- **MayMoveOnClick** (enum PeterBlum.DES.TrueFalseIgnore) – Determines if the buttons in DetailViews and GridViews may move if clicked when focus is in an editable control whose value changed, causing a validator to appear or disappear. Other DataBound controls explicitly declare their buttons and can setup individual <des:Button> tags with **MayMoveOnClick** explicitly definied.

  This property has these values:

  - `True` – Set **MayMoveOnClick** to `true`.
  - `False` – Set **MayMoveOnClick** to `false`, overriding any existing setting of true on the control.
  - `Ignore` – Use the existing property setting on the control. This is the default.

- **ChangeMonitorEnables** (enum PeterBlum.DES.ChangeMonitorEnablesSubmitControl) – Determines if the ChangeMonitor handles enabling and disabling this submit control as the changemonitor group is either set or cleared.

  Typically this is used on buttons whose **CausesValidation** property is `true` so that a Cancel button will always be active.

  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. This is the default.
  - `CausesValidationIsFalse` - When the button's **CausesValidation** property is `false`, it will change its enabled state.

- **DeleteConfirmMessage** (string) – Displays this text as a confirm message on the Delete button that is setup by a CommandField or ButtonField whose **CommandName**=`Delete`. When "", no confirm message is shown.

  It defaults to "`Are you sure you want to delete this item?`".

  This string is available for use in button controls that you add to the page like this:

```
<des:Button ConfirmMessage=
     "<%# PeterBlum.DES.DynamicData.Util.GetDeleteConfirmMessage() %>" />
```

- **UpdateConfirmMessage** (string) – Displays this text as a confirm message on the Update button that is setup by a CommandField or ButtonField whose **CommandName**=`Update`. When "", no confirm message is shown.

  It defaults to "".

  This string is available for use in button controls that you add to the page like this:

```
<des:Button ConfirmMessage=
     "<%# PeterBlum.DES.DynamicData.Util.GetUpdateConfirmMessage() %>" />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **CancelConfirmMessage** (string) – Displays this text as a confirm message on the Cancel button that is setup by a CommandField or ButtonField whose **CommandName**=`Cancel`. When "", no confirm message is shown.

  It defaults to "".

  This string is available for use in button controls that you add to the page like this:

```
<des:Button ConfirmMessage=
    "<%# PeterBlum.DES.DynamicData.Util.GetCancelConfirmMessage() %>" />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Validation Properties

Note: If you want to customize the ErrorFormatter used on the validators, add a PageManager control and use the DefaultErrorFormatterSkinID property. See the Validation User's Guide for details.

- **ErrorMessages** (System.Collections.Generic.List<PeterBlum.DES.DynamicData.ValidatorErrorMessages>) – A list of ValidatorErrorMessage definitions where the user maps a type of validator to its default error messages.

  This is where you establish ErrorMessage and SummaryErrorMessage strings that override the values determined on the ValidationAttributes. After all, ValidationAttributes are established in the database layer and error messages are really user interface specific.

  Each `PeterBlum.DES.DynamicData.ValidationErrorMessage` class covers all validators on the page of a specific type, such as RequiredTextValidator and RangeValidator. In addition, it can look at the column's datatype, as determined by any DataTypeAttribute on the column or the actual type of the column. For example, you can have separate ValidationErrorMessages objects for RangeValidators on DateTime and Integer columns.

Properties of the ValidationErrorMessage class

- **ValidatorType** (Type) – The class type to match to a DES Validator class. If this type is not a subclass of PeterBlum.DES.BaseAnyValidator, an exception is thrown.

  This property can only be assigned programmatically. Use NameOfValidatorType (below) to assign it in the ASP.NET Declarative Syntax.

  [C#]

  ```
  vValidatorErrorMessage.ValidatorType = typeof(PeterBlum.DES.RequiredTextValidator);
  ```

  [VB]

  ```
  vValidatorErrorMessage.ValidatorType = GetType(PeterBlum.DES.RequiredTextValidator)
  ```

- **NameOfValidatorType** (string) – A string representing the type of validator. When the type is defined in PeterBlum.DES, its just the actual name of the type. If its a full type name, then it will look in the current web application assembly. If its an assembly qualified name, it will look in the associated assembly.

  There are 3 formats supported:

  - If its in the PeterBlum.DES assembly, just use the class name. It should not specify the namespace. For example: `"CharacterSetValidator"`.
  - If its in the calling web application assembly, specify the full name of the type which is namespace + class. For example: `"PeterBlum.Custom.MyValidator"`.
  - For all other cases, use the assembly qualified name which is a format defined in the .net framework like this: Full name, assemblyname [optional culture, version, and publickeytoken]. For example: `"PeterBlum.Custom.MyValidator, MyAssembly, Culture=en, PublicKeyToken=b5d015c7a5a0b013, Version=1.0.0.0 "`.

  You must either set **NameOfValidatorType** or **ValidatorType** properties.

- **DataTypeName** (string) – You can refine the mapping of a ValidatorErrorMessage to a validator by matching to the datatype of the column. The datatype is determined from any DataTypeAttribute assigned to the column. If it lacks a DataTypeAttribute, it uses the actual property type on the EntityTable Class. This is the string name of the datatype.

  When unassigned, it is ignored. Otherwise the name must match one of these sources:

  - System.Web.DynamicData.DataTypeAttribute.DataType property. Used when DataTypeAttribute is applied to the column. For example, when DataTypeAttribute.DataType=DataType.Date, you will assign "Date".
  - System.Web.DynamicData.DataTypeAttribute.CustomType property. Used when DataTypeAttribute is applied to the column and has been set to a string.
  - PeteBlum.DES.DynamicData.DataTypeAttribute.ExtendedDataType property. Used when DES DataTypeAttribute is applied to the column.
  - The name of a real type from this list: Integer, Double, Decimal, String, Boolean, DateTime.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

It is case insensitive.

It defaults to "".

- **ErrorMessage** (string) – Text displayed at the validator's location on the page through its ErrorFormatter. This value is copied to the Validator's **ErrorMessage** property.

  Supports the same tokens found on the validator created by the ValidationAttribute. For example, when using the CompareToValueAttribute, it creates the CompareToValueValidator which offers the token "{VALUETOCOMPARE}".

  The token "{LABEL}" is always available to show the column name. (The DisplayNameAttribute can be used to create a user friendly name.)

  Note: ASP.NET Dynamic Data supports tokens in the ValidationAttribute. They differ from those used by DES, such as {0} for the label. DES supports most of those tokens but its recommended you use DES's tokens for clarity.

- **ErrorMessageLookupID** (string) – Gets the value for **ErrorMessage** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of ErrorMessage. If no match is found OR this is blank, **ErrorMessage** 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.

- **SummaryErrorMessage** (string) – Supplies the error message for the ValidationSummary control and the alert shown when using **PeterBlum.DES.Globals.Page.ShowAlertOnSubmit**. You can leave it blank if the value of **ErrorMessage** is the message you want in the ValidationSummary and alert.

  SummaryErrorMessage supports the same tokens and editor in the Properties Editor as **ErrorMessage** does.

- **SummaryErrorMessageLookupID** (string) – Gets the value for **SummaryErrorMessage** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of ErrorMessage. If no match is found OR this is blank, **SummaryErrorMessage** 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.

Constructors for PeterBlum.DES.DynamicData.ValidationErrorMessage

The parameters match properties shown in the previous section.

[C#]

```
ValidatorErrorMessages()
ValidatorErrorMessages(Type ValidatorType, string ErrorMessage)
ValidatorErrorMessages(Type ValidatorType, string DataTypeName,
   string ErrorMessage)
ValidatorErrorMessages(Type ValidatorType, string DataTypeName,
   string ErrorMessage, string SummaryErrorMessage)
```

[VB]

```
ValidatorErrorMessages()
ValidatorErrorMessages(ByVal ValidatorType As Type,
   ByVal ErrorMessage As String)
ValidatorErrorMessages(ByVal ValidatorType As Type,
   ByVal DataTypeName As String,
   ByVal ErrorMessage As String)
ValidatorErrorMessages(ByVal ValidatorType As Type,
   ByVal DataTypeName As String,
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

```
        ByVal ErrorMessage As String, ByVal SummaryErrorMessage As String)
```

Adding ValidationErrorMessage objects in ASP.NET Declarative Syntax

```
<desDD:Customizer ID="Customizer1" runat="server">
   <ErrorMessages>
      <desDD:ValidatorErrorMessages NameOfValidatorType="RequiredTextValidator"
         ErrorMessage="{LABEL} requires a value."
      />
      <desDD:ValidatorErrorMessages NameOfValidatorType="RangeValidator"
         DataTypeName="Integer"
         ErrorMessage="Use the range of {MINIMUM} to {MAXIMUM}."
         SummaryErrorMessage="Use the range of {MINIMUM} to {MAXIMUM} on {LABEL}."
      />
   </ErrorMessages>
</desDD:Customizer>
```

Adding ValidationErrorMessage objects programmatically

This code reproduces the previous example programmatically.

[C#]

```
Customizer1.ErrorMessages.Add(
   new PeterBlum.DES.DynamicData.ValidationErrorMessage(
      typeof(PeterBlum.DES.RequiredTextValidator),
      "{LABEL} requires a value."));
Customizer1.ErrorMessages.Add(
   new PeterBlum.DES.DynamicData.ValidationErrorMessage(
      typeof(PeterBlum.DES.RangeTextValidator), "Integer",
      "Use the range of {MINIMUM} to {MAXIMUM}.",
      "Use the range of {MINIMUM} to {MAXIMUM} on {LABEL}."));
```

[VB]

```
Customizer1.ErrorMessages.Add( _
   New PeterBlum.DES.DynamicData.ValidationErrorMessage( _
      GetType(PeterBlum.DES.RequiredTextValidator), _
      "{LABEL} requires a value."))
Customizer1.ErrorMessages.Add( _
   New PeterBlum.DES.DynamicData.ValidationErrorMessage( _
      GetType(PeterBlum.DES.RangeTextValidator), "Integer", _
      "Use the range of {MINIMUM} to {MAXIMUM}.", _
      "Use the range of {MINIMUM} to {MAXIMUM} on {LABEL}."))
```

- **ColumnValidatorManager_DisplayMode** (enum PeterBlum.DES.DynamicData. ColumnValidatorManager_DisplayMode) – Determines the appearance of inline validation error messages on ColumnValidatorManager or CombinedErrorMessages controls of a Field Template.

  The enumerated type `PeterBlum.DES.DynamicData.ColumnValidatorManager_DisplayMode` has these values:

  - `Default` – Use the current setting of the ColumnValidatorManager's **ListStyle** property. This is the default.
  - `None` – Hide the inline error messages. Only show errors in the ValidationSummary control. Effectively sets **ColumnValidatorManager.ErrorFormatter.Display** to `None`.
  - `Paragraph` – Show the error messages in a paragraph style with the value **ColumnValidatorManager_ParagraphSeparator** used to separate messages. Effectively sets **ColumnValidatorManager.ListStyle** to `false`.
  - `List` – Show the error messages in a list style with the value **ColumnValidatorManager_ListLeadText** preceding each message. Effectively sets **ColumnValidatorManager.ListStyle** to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **ColumnValidatorManager_ListLeadText** (string) – Used when **ColumnValidatorManager_DisplayMode** = `List` as the text that precedes each error message.

  When the token {DEFAULT} is used, it does not change the current setting of these controls.

  It defaults to `"{DEFAULT}"`.

- **ColumnValidatorManager_ParagraphSeparator** (string) – Used when **ColumnValidatorManager_DisplayMode** = `Paragraph` as the text that separates each error message.

  When the token {DEFAULT} is used, it does not change the current setting of these controls.

  It defaults to `"{DEFAULT}"`.

- **ColumnValidatorManager_ShowRequiredFieldMarker** (enum PeterBlum.DES.TrueFalseIgnore) – Adds a RequiredFieldMarker control to the left of the error message on any required field.

  Only assigned to a CombinedErrorMessages or ColumnValidatorManager control on the Field Template, not individual validators. Assigned when the MetaColumn indicates its required.

  Use it as a shortcut to setting up the RequiredFieldMarker.

  This property has these values:

  - `True` – Set **ShowRequiredFieldMarker** to `true`.
  - `False` – Set **ShowRequiredFieldMarker** to `false`, overriding any existing setting of `true` on the control.
  - `Ignore` – Use the existing property setting on the control. This is the default.

- **ColumnValidatorManager_NoErrorFormatterMode** (enum PeterBlum.DES.NoErrorFormatterMode) – Determines the Mode property on the ColumnValidatorManager's NoErrorFormatter property which shows an indicator when there is no error message shown.

  The enumerated type `PeterBlum.DES.NoErrorFormatterMode` has these values:

  - `Off` – Do not use the **NoErrorFormatter**.
  - `Default` – Use the value assigned to the **NoErrorFormatter.Mode** property on the ColumnValidatorManager control assigned to the Field Template. If that property is set to Default, it will use the global setting from the **DefaultNoErrorFormatterMode** property in the **Global Settings Editor**.
  - `Always` – Always show when there is no error. It will appear as the page is first displayed. It will be removed as an error is revealed. It will return after the error is fixed.
  - `Corrected` – Show after the user fixes the error. It appears after the Validator displays the error and the user corrects it.
  - `Validated` – Show after the field has been validated and has no errors. The user does not have to edit to cause this. For example, if a field requires text and opens with text in the textbox, when the user clicks Submit, the RequiredTextValidator will validate and the **NoErrorFormatter** appears.
  - `FirstTime` - Show only on a new page, prior to any edits. It is hidden after validation occurs. Never show on a post back page because validation has already occurred. It lets you attract users to fields that have not been edited yet.

  It defaults `NoErrorFormatterMode.Default`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

ForeignKey Field Template Properties

- **ForeignKeyRequireSelectionText** (string) – When the MetaColumn requires a value, ForeignKeys use this as the first element of their lists.

  Individual ForeignKey Field Templates can override this by changing their **RequireSelectionText** property from "{DEFAULT}" to the desired text.

  It defaults to `"[Make a Selection]"`.

  Supported by any Field Template subclassed from PeterBlum.DES.DynamicData.BaseForeignKeyEditFTUC, which includes **ForeignKey_Edit.ascx**.

- **ForeignKeyRequireSelectionTextLookupID** (string) – An alternative to **ForeignKeyRequireSelectionText**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. Strings must be in the string group Misc.

- **ForeignKeyNoSelectionText** (string) – When the MetaColumn does not require a value, ForeignKeys use this as the first element of their lists.

  Individual ForeignKey Field Templates can override this by changing their **NoSelectionText** property from "{DEFAULT}" to the desired text.

  It defaults to `"[Not Set]"`.

  Supported by any Field Template subclassed from PeterBlum.DES.DynamicData.BaseForeignKeyEditFTUC, which includes **ForeignKey_Edit.ascx**.

- **ForeignKeyNoSelectionTextLookupID** (string) – An alternative to **ForeignKeyNoSelectionText**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. Strings must be in the string group Misc.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Miscellaneous Properties

- **ReadOnlyUI** (enum PeterBlum.DES.DynamicData.ReadOnlyUI) – Determines the default user interface when an editable Field Template needs to switch to a ReadOnly interface. It can show either a label or disable the edit control.

  The enumerated type `PeterBlum.DES.DynamicData.ReadOnlyUI` has these values:

  - `Disable` – Disable the edit control. This is the default.
  - `Label` – Remove the edit control. Show a label in its place showing the value.

- **DependencyUI** (enum PeterBlum.DES.DynamicData.DependencyUI) – When using a DependencyAttribute, a FieldStateController is optionally setup to change the appearance of the data entry control when the dependency does not need the column to be assigned a value.

  The enumerated type `PeterBlum.DES.DynamicData.DependencyUI` has these values:

  - `None` – Do not change the appearance. The control is always editable.
  - `Disable` – Disable the edit control. This is the default.
  - `Hide` – Hide the edit control.
  - `ReadOnlyOrDisable` – When using a textbox, set it to ReadOnly mode. For all other controls, disable.

- **UseDependencyAttributes** (boolean) – When a MetaColumn has a DependencyAttribute assigned, it will disable the column's data entry and validators at times. If you don't want that behavior, set this to `false`.

  The Field Template has a way to override this with its own **UseDependencyAttribute** property. The Customizer's **UseDependencyAttribute** will only affect the Field Template when its own **UseDependencyAttribute** is set to `TrueFalseDefault.Default`.

  When `true`, the DependencyAttribute is used.

  When `false`, it is not. The data entry control is always active along with its validators.

  It defaults to `true`.

- **ApplyUIHintControlParameters** (enum PeterBlum.DES.DynamicData.ApplyUIHintControlParameters) – Determines if and when a UIHintAttribute applies its ControlParameters to the properties of this Field Template. When applied early, you can overwrite their settings in `Page_Load()`. When applied late, they will overwrite your settings in PreRender.

  The Field Template has a way to override this with its own **ApplyUIHintControlParameters** property. The Customizer's **ApplyUIHintControlParameters** will only affect the Field Template when its own **ApplyUIHintControlParameters** is set to `null`/`Nothing`.

  The enumerated type `PeterBlum.DES.DynamicData.ApplyUIHintControlParameters` has these values:

  - `No` – Do not apply them.
  - `First` – Apply them early (in Page_Init) so they can be overridden in `Page_Load()`. This is the default.
  - `Last` – Apply them late, overridding the values that may have come from other sources.

- **UseDescriptionAsToolTipReadOnly** (Boolean) – When the Field Template is shown in ReadOnly mode, the MetaColumn.Description value can be assigned to the ToolTip of the DataControl.

  ReadOnly is determined by the **MetaColumn.IsReadOnly** property and when the Field Template's **Mode** property is set to `ReadOnly`.

  The Field Template has a way to override this with its own **UseDescriptionAsToolTip** property. The Customizer's **UseDescriptionAsToolTipReadOnly** will only affect the Field Template when its own **UseDescriptionAsToolTip** is set to `TrueFalseDefault.Default`.

  It defaults to `false`.

- **UseDescriptionAsToolTipEdit** (Boolean) – When the Field Template is shown in Edit mode, the MetaColumn.Description value can be assigned to the ToolTip of the DataControl.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Edit is determined by the Field Template's **Mode** property when set to `Edit`.

The Field Template has a way to override this with its own **UseDescriptionAsToolTip** property. The Customizer's **UseDescriptionAsToolTipEdit** will only affect the Field Template when its own **UseDescriptionAsToolTip** is set to `TrueFalseDefault.Default`.

It defaults to `false`.

- **UseDescriptionAsToolTipInsert** (Boolean) – When the Field Template is shown in Insert mode, the MetaColumn.Description value can be assigned to the ToolTip of the DataControl.

  Insert is determined by the Field Template's **Mode** property when set to `Insert`.

  The Field Template has a way to override this with its own **UseDescriptionAsToolTip** property. The Customizer's **UseDescriptionAsToolTipInsert** will only affect the Field Template when its own **UseDescriptionAsToolTip** is set to `TrueFalseDefault.Default`.

  It defaults to `false`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

ColumnValidatorManager Control

The ColumnValidatorManager control creates the Validators on a Field Template automatically so you don't have to add Validators to the Field Templates. Instead, you add one ColumnValidatorManager control in the location where you want the validators to report their errors (such as just after the data entry control). Since it's a subclass of the DES CombinedErrorMessages control, it knows how to format error messages from multiple validators. It also uses the **DisplayOrder** property on ValidationAttributes to dictate the order they are shown.

It gets the information to create its validators from these sources:

- ValidationAttributes and DataTypeAttributes – Either assigned on the EntityTable MetaData classes as attributes or assigned at runtime using the ICustomizeColumn and IUICustomizeColumn interfaces.
- ICreateValidatorWebControls interface – Creates any other validators that could not be described as attributes.
- Customizer control - Its **ErrorMessages** property provides values for the **ErrorMessage**, **SummaryErrorMessage**, **ErrorMessageLookupID** and **SummaryErrorMessageLookupID** properties.

It automatically creates Required, DataTypeCheck and the DynamicValidator when needed but not assigned in the metadata. This will further protect you from creating templates that allow data to bypass the business logic.

Click on any of these topics to jump to them:

- Setting Appearance properties
- Modifying Validators after they have been created

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Setting Appearance properties

You can assign properties like ErrorFormatter and NoErrorFormatter directly on the Field Templates. However, that takes a lot of work and some Field Templates may accidentally be missed. Instead use DES's page-level settings found on its PageManager and Customizer controls.

Setting the default ErrorFormatter

Use the **DefaultErrorFormatterSkinID** property on the PageManager control to determine the ErrorFormatter type and some of its properties. The DefaultErrorFormatterSkinID property has a special syntax, described in the Validation User's Guide, that lets you state the name of an ErrorFormatter followed by a list of properties and values.

This example shows the **DefaultErrorFormatterSkinID** using the PopupErrorFormatter with its Display property set to Static.

```
<des:PageManager ID="DESPageManager" runat="server"
   AJAXFramework="MicrosoftAJAX"
   DefaultErrorFormatterSkinID="PopupView {Display='Dynamic' }" />
```

Showing the RequiredFieldMarker

Use the **ColumnValidatorManager_ShowRequiredFieldMarker** property on the Customizer control to show or hide the RequiredFieldMarker.

```
<desDD:Customizer ID="Customizer1" runat="server"
   ColumnValidatorManager_ShowRequiredFieldMarker="True" />
```

Note: The tagPrefix "desDD" refers to the PeterBlum.DES.DynamicData assembly and namespace.

Setting the DisplayMode for when multiple ErrorMessages are shown

When more than one validator reports an error at the same time, their error messages will be combined. Use the **ColumnValidatorManager_DisplayMode** property on the CustomizerControl to determine if they are combined into a paragraph style or a list style. Use the **ColumnValidatorManager_ListLeadText** property to further format the list style and **ColumnValidatorManager_ParagraphSeparator** to further format the paragraph style.

```
<desDD:Customizer ID="Customizer1" runat="server"
   ColumnValidatorManager_DisplayMode="List"
   ColumnValidatorManager_ListLeadText="+ " />
```

```
<desDD:Customizer ID="Customizer1" runat="server"
   ColumnValidatorManager_DisplayMode="Paragraph"
   ColumnValidatorManager_ParagraphSeparator=" | " />
```

Setting the NoErrorFormatter mode

Use the **ColumnValidatorManager_NoErrorFormatterMode** property on the Customizer control use the NoErrorFormatter feature. See the Validation User's Guide for details on this feature.

```
<desDD:Customizer ID="Customizer1" runat="server"
   ColumnValidatorManager_NoErrorFormatterMode="Validated" />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Modifying Validators after they have been created

You can use the ICustomizeFieldTemplate interface to modify a Field Template's appearance after all of its controls have been created and registered. Its `CustomizeFieldTemplate()` method is an excellent place to modify individual validators, such as changing their ErrorFormatter or ErrorMessage.

Within `CustomizeFieldTemplate()`, use the `FindValidator()` method on the *pArgs* parameter to retrieve a validator control by specifying its type. Then modify its propertise. See "FindValidator Method".

Example

Changes the **ErrorFormatter** property to HyperLinkErrorFormatter on any DataTypeCheckValidator.

[C#]

```
public void CustomizeFieldTemplate(
   DES.DESFieldTemplateUserControl pFieldTemplate,
   DES.CustomizeFieldTemplateEventArgs pArgs)
{
   PeterBlum.DES.BaseAnyValidator vValidator =
      pArgs.FindValidator(typeof(PeterBlum.DES.DataTypeCheckValidator));
   if (vValidator != null)
      vValidator.ErrorFormatter = new HyperLinkErrorFormatter();
}
```

[VB]

```
Public Sub CustomizeFieldTemplate( _
   ByVal pFieldTemplate As DES.DESFieldTemplateUserControl, _
   ByVal pArgs As DES.CustomizeFieldTemplateEventArgs) _
  Implements DES.ICustomizeFieldTemplate.CustomizeFieldTemplate

   Dim vValidator As PeterBlum.DES.BaseAnyValidator = _
      pArgs.FindValidator(GetType(PeterBlum.DES.DataTypeCheckValidator))
   If Not vValidator Is Nothing Then
      vValidator.ErrorFormatter = New HyperLinkErrorFormatter()
   End If
End Sub
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

DynamicValidator Control

ASP.NET Dynamic Data allows you to create validation code within the EntityTable class. If your code detects an error, it should throw a `System.ComponentModel.DataAnnotations.ValidationException` with an error message. The DynamicValidator control handles that exception, displaying the error message in a validator format.

When using DES, you must use its replacement to the native DynamicValidator control, `PeterBlum.DES.DynamicData.DynamicValidator`.

This control is automatically created for you when you use the ColumnValidatorManager and DES DynamicDataManager control. If you do not use the ColumnValidatorManager on your Field Template or DES DynamicDataManager on your Page Template, add this to your Field Template or Page Template:

```
<desDD:DynamicValidator id="DynamicValidator1" runat="server" />
```

Note: The tagPrefix "desDD" refers to the PeterBlum.DES.DynamicData assembly and namespace.

In the Field Template's `Page_Init()` method, register it by calling `SetUpDESDynamicValidator(DynamicValidator1)`.

See "Writing server side only Validation Code within your EntityTable Classes".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

The DynamicLabel Control

Note: This is not the same control or a subclass of the control found in Dynamic Data Futures. It implements the same features though.

The native Label control is often used to hold the text of a label to a data entry control, such as a TextBox. When building data oriented web forms, the label is usually the column name or from either the DisplayNameAttribute or DisplayAttribute. It can also hold a description or even a prompt string, where the text comes from either the DescriptionAttribute or DisplayAttribute.

The DynamicLabel is a subclass of the Label control (actually of DES's LocalizableLabel) that includes these extensions:

- It identifies the column whose label is shown from either the **DataField** or **AssociatedControlID** property. When using **AssociatedControlID**, point it to a DynamicControl and it will use the **DataField** value from that DynamicControl.
- It can define its text from any of the sources: display name, short display name, description, prompt and other.
- The column name can be embedded within the **Text** property using a "{COLUMN}" token and can have its case changed in several ways. If you want to show a different column name in the text, use the {COLUMN:column name} token.

```
Text="The column named {COLUMN}."
```

You may find it easier to just declare the text of the column name directly in the templates of your FormView or ListView controls. DynamicLabel becomes more important when:

- You are separating the user interface from the business logic and have determined the business logic as the source for column names. (It is already the source for GridView column titles and DetailsView row titles.) This way, a name change in the business logic is reflected automatically.
- You create a control designed for reuse, where it contains DynamicControls and DynamicLabels that do not have a predefined column name in their **DataField** properties. Instead, your control provides properties where the column names are declared and passed into the DynamicControls and DynamicLabels.

Click on any of these topics to jump to them:

- Using the DynamicLabel Control
- Properties on the DynamicLabel Control

```
<desDD:DynamicLabel id="DynamicLabel1" runat="server" DataField="PhoneNumber" />
<desDD:DynamicLabel id="DynamicLabel2" runat="server" DataField="PhoneNumber"
  TextSource="Description" />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the DynamicLabel Control

Click on any of these topics to jump to them:

- Establishing Metadata on a column
- Establishing the DataField name
- Choosing a source for the column's metadata
- Formatting the text
- Enclosing the label in other HTML formatting

Establishing Metadata on a column

Use the DisplayNameAttribute to define a user friendly column name and an optional shorter version of it.

[C#]

```
[DES.DisplayName("Start Date", "", "Start", "")]
public object StartDate { get; set; }
```

[VB]

```
<DES.DisplayName("Start Date", "", "Start", "") > _
Public Property StartDate As Object
  Get
    Return Nothing
  End Get
  Set
  End Set
End Property
```

Use the DescriptionAttribute to define two forms of descriptions that can appear in the user interface called "Description" and "Prompt". The "description" is automatically used in tooltips within Field Templates.

[C#]

```
[DES.Description("The date when the project started.", "",
  "Enter the date when the project started.", "")]
public object StartDate { get; set; }
```

[VB]

```
<DES.Description("The date when the project started.", "", _
  "Enter the date when the project started.", "") > _
Public Property StartDate As Object
  Get
    Return Nothing
  End Get
  Set
  End Set
End Property
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Establishing the DataField name

The DynamicLabel works like an ordinary Label control until you want to display text associated with a column's metadata. To use the column's metadata, assign the column name in the **DataField** property or point the **AssociatedControlID** to a DynamicControl.

When used within a Pattern Template, you can leave the **DataField** property unassigned to let the DynamicPattern, DynamicListView, or DynamicFormView control determine its value.

Exceptions are thrown at runtime when column names specified in the **DataField** property are unknown.

Example: Column name from DataField

Note: The tagPrefix "desDD" refers to the PeterBlum.DES.DynamicData assembly and namespace.

```
<desDD:DynamicLabel id="DynamicLabel1" runat="server" DataField="PhoneNumber" />
```

Example: Column name from AssociatedControlID

Note: The tagPrefix "desDD" refers to the PeterBlum.DES.DynamicData assembly and namespace.

```
<desDD:DynamicLabel id="DynamicLabel2" runat="server"
  AssociatedControlID="PhoneNumberDynamicControl" /> 
<desDD:DynamicControl id="PhoneNumberDynamicControl" runat="server"
  DataField="PhoneNumber" />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Choosing a source for the column's metadata

There are several sources for text associated with a column. They come from either the DisplayNameAttribute, DescriptionAttribute, or DisplayAttribute. Specify which to use within the **TextSource** and **TextSourceFallback** properties of the DynamicLabel control.

- The **DisplayName** property, defined either by with DisplayNameAttribute or DisplayAttribute.
- The **ShortDisplayName** property, defined either by with DisplayNameAttribute or DisplayAttribute.
- The **Description** property, defined either by with DescriptionAttribute or DisplayAttribute.
- The **Prompt** property, defined either by with DescriptionAttribute or DisplayAttribute.
- The **Other** property of the Label control itself.

Example: Column's description

Note: The tagPrefix "desDD" refers to the PeterBlum.DES.DynamicData assembly and namespace.

```
<desDD:DynamicLabel id="DynamicLabel3" runat="server" DataField="PhoneNumber"
  TextSource="Description" />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Formatting the text

The **Text** property determines if the column's text appears and if so, where. Whenever it appears, the column's text is determined by the **TextSource** and **TextSourceFallback** properties as described above.

- If the **Text** property is blank, it will be assigned the column's text.
- If the **Text** property is not blank, the "{COLUMN}" token is replaced by the column's text. For example:

```
Text="{COLUMN}:"
```

- If the **Text** property contains "{COLUMN:*column name*}" tokens, they are replaced by the column's text defined in the token. For example:

```
Text="{COLUMN} through {COLUMN:enddate}"
```

- If the TextLookupID property is assigned, it gets the initial value for the Text property from the String Lookup System (see the **General Features Guide**.)

Use the **Case** property to change the case of its characters.

Use the **CssClass** property to specify a style sheet class name.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Enclosing the label in other HTML formatting

When using a Pattern Template, sometimes the DynamicLabel is present but may not be used. It usually is surrounded by HTML formatting, such as `[DynamicLabel] [DynamicControl]`. If the DynamicLabel isn't shown, you may want to hide that formatting. DynamicLabel makes it easy with its **HTMLBefore** and **HTMLAfter** properties. Move the HTML into those tags and it will know to omit that HTML when the main text of the label isn't shown.

Example: Using HTMLAfter

Compare this to the previous example. The ` ` that separates the two controls is now within the **HTMLAfter** property.

Note: The tagPrefix "desDD" refers to the PeterBlum.DES.DynamicData assembly and namespace.

```
<desDD:DynamicLabel id="DynamicLabel2" runat="server"
   AssociatedControlID="PhoneNumberDynamicControl" HTMLAfter="&amp;nbsp;" />
<desDD:DynamicControl id="PhoneNumberDynamicControl" runat="server"
   DataField="PhoneNumber" />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties on the DynamicLabel Control

Click on any of these topics to jump to them:

- Source of data Properties
- Textual Content Properties
- Additional formatting properties

Source of data Properties

- **DataField** (string) – Column name where the column name will be looked up. It looks for a matching MetaColumn with this name and uses its DisplayNameAttribute. If no such attribute is used, it uses the name of the property in the EntityTable class.

  If this is blank, the column name can still be retrieved from the DynamicControl assigned to the **AssociateControlID** property.

  DataField is not used when the **Text** property is assigned but does not have a "{COLUMN}" token. If it has a "{COLUMN}" token, this property determines its value.

  It defaults to "".

- **AssociatedControlID** (string) – An ID to another control associated with this label. When assigned to a DynamicControl, it will actually use the data entry control created by the DynamicControl. In addition, if the DataField property is blank, it uses the DynamicControl's own DataField to determine the column name.

  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.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Textual Content Properties

- **Text** (string) – It supplies the text shown by the control. It supports HTML tags. It may be replaced by the value from TextLookupID or by the column name. It supports tokens and column name lookup as described in "Using the DynamicLabel Control".

  It defaults to "".

- **TextLookupID** (string) – Gets the value for **Text** through the String Lookup System (see the **General Features Guide**). 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 "".

- **TextSource** (enum PeterBlum.DES.DynamicData.ColumnTextSource) – Determines which of element in the column's metadata is shown.

  If the textual value found is unassigned, it will use the TextSourceFallback property to find an alternative. When TextSource=DisplayName or ShortDisplayName, it will never need to use the TextSourceFallback as MetaColumn internally falls back from ShortDisplayName to DisplayName to the actual column name.

  The enumerated type `PeterBlum.DES.DynamicData.ColumnTextSource` has these values:

  - `DisplayName` – Uses the **DisplayName** property, defined either by with DisplayNameAttribute or DisplayAttribute.

    If DisplayNameAttribute or DisplayAttribute do not supply a value, this will always fallback to using the column actual name.

  - `ShortDisplayName` – Uses the **ShortDisplayName** property, defined either by with DisplayNameAttribute or DisplayAttribute.

    If DisplayNameAttribute or DisplayAttribute do not supply a value, this will always fallback to using the DisplayName.

  - `Description` – Uses the **Description** property, defined either by with DescriptionAttribute or DisplayAttribute. If unassigned when using **TextSource**=`Description`, it will use **TextSourceFallback** to get an alternative.

    Note: While the FieldTemplateUserControl has access to the customized value in ModifiableColumn.Description, the DynamicLabel does not. The DynamicLabel will never show the customized description.

  - `Prompt` – Uses the **Prompt** property, defined either by with DescriptionAttribute or DisplayAttribute. If unassigned when using **TextSource**=`Prompt`, it will use **TextSourceFallback** to get an alternative.

  - `Other` – Uses the values of the OtherText and OtherTextLookupID properties, defined on the DynamicLabel control. If unassigned when using **TextSource**=`Other`, it will use **TextSourceFallback** to get an alternative.

  It defaults to `ColumnTextSource.DisplayName`.

- **TextSourceFallback** (enum PeterBlum.DES.DynamicData.ColumnTextSource) – Used when TextSource=Description, Prompt, or Other and an empty string was returned. This is used instead. See the TextSource property for details on its values.

  It defaults to `ColumnTextSource.Other`.

- **OtherText** (string) – When **TextSource** or **TextSourceFallback** specify `Other`, this property contains the text or HTML to use.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **OtherTextLookupID** (string) – Gets the value for **OtherText** through the String Lookup System (see the **General Features Guide**). The LookupID and its value should be defined within the String Group of Labels. If no match is found OR this is blank, **OtherText** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Additional formatting properties

- **CssClass** (string) – The style sheet class name to apply to the HTML. It is unassigned by default.
- **Case** (enum PeterBlum.DES.CaseConversion) – Adjusts the case of the column's text. The enumerated type `PeterBlum.DES.CaseConversion` has these values:
  - `None` – no conversion. This is the default.
  - `Lowercase`
  - `Uppercase`
  - `SentenceCase` – Only the first letter is capitalized
  - `TitleCase` – The first letter of each word is capitalized
- **HTMLBefore** (string) – Include HTML that appears before the label. It is **not** shown if the label is empty. This allows a label that is intended to show a column to be completely blank when no column name is assigned.

  This HTML is inside the `<span>` tag that contains the label. So it inherits the styles applied to the label.

  It defaults to "".
- **HTMLAfter** (string) – Include HTML that appears after the label. It is **not** shown if the label is empty. This allows a label that is intended to show a column to be completely blank when no column name is assigned.

  This HTML is inside the `<span>` tag that contains the label. So it inherits the styles applied to the label.

  It defaults to "".
- **HTMLBeforeLookupID** (string) – Gets the value for **HTMLBefore** through the String Lookup System (see the **General Features Guide**). The LookupID and its value should be defined within the String Group of Labels. If no match is found OR this is blank, **HTMLBefore** 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 "".
- **HTMLAfterLookupID** (string) – Gets the value for **HTMLAfter** through the String Lookup System (see the **General Features Guide**). The LookupID and its value should be defined within the String Group of Labels. If no match is found OR this is blank, **HTMLAfter** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

The DynamicButtons Control

The DynamicButtons control is much like the CommandField class, a control you use within the GridView and DetailsView controls to offer buttons such as New, Edit, Save and Cancel. These buttons are invoke commands on a DataBound control. Some, like New and Edit switch data entry modes. Others, like Save and Cancel finish a data entry mode.

The DynamicButtons control has these features:

- Describe buttons using properties or Templates. With Templates, you have complete control over the HTML and web controls that are output. With properties, it will create your choice of button type (Button, LinkButton, or ImageButton), applying text and image URLs from properties. You can even rearrange the order of the buttons and change the HTML between them using "Layout" properties.
- Offers different buttons based on the data entry mode.
  - For ReadOnly mode on a list-style DataBound control, it offers: Edit (start Edit mode), Delete (prompt then delete a record), Select (mark the row as selected), and Details (view the row in a single record view.)
  - For ReadOnly mode on a single record-style DataBound control, it offers: New (start Insert mode), Edit (start Edit mode), and Delete (prompt then delete the record).
  - For Edit and Insert modes, it offers: Save and Cancel.
- Use information from the DataSource and DataBound Control Adapter to determine certain buttons are hidden. For example, the Delete button is hidden DataSource.EnableDelete is `false`.
- Optionally setup a containing `<span>` or `<div>` tag around all buttons with a style sheet class.
- Apply properties from the Customizer Control, such as **DisableOnSubmit**, **ChangeMonitorEnables**, and various **ConfirmMessages**.
- Direct the ENTER key to click the Save button when hit in a data entry field within the DataBound control.
- Unlike the CommandField, it is a standard web control and can be placed within Templates on ListView and FormView.

Click on any of these topics to jump to them:

- Using the DynamicButtons Control
- Adding a DynamicButtons Control
- Properties for the DynamicButtons Control
- The VisibilityByCommand Control

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the DynamicButtons Control

The DynamicButtons control is subclassed from `System.Web.UI.WebControls.WebControl`. It can be added into any DataBound control that is attached to Dynamic Data through the DynamicDataManager control.

It should be placed in a Template. Within ListView and DetailsView, it goes into ItemTemplate, EditItemTemplate, etc. Within GridView or DetailsView, it goes into a TemplateField object. When using a DynamicPattern, DynamicListView, or DynamicFormView control, it can go into a Pattern Template file or a TemplateNotInPattern object.

It can also be placed within a DynamicControlPanel control, which is outside of the actual DataBound control, by using the DynamicButtonsField in the Fields collection of the DynamicControlPanel. When used this way with a list-style DataBound control, like DynamicListView, it switches to the buttons offered for a single record DataBound control: New, Edit, and Delete. It also allows you to add the Details button. The Edit, Delete, and Details buttons depend on if there is a selected row in the list. For details, see "Using the DynamicButtonsField".

Click on any of these topics to jump to them:

- Choosing the Appearance: from Properties or Templates
- Using the Layout Properties
- Using Templates

Choosing the Appearance: from Properties or Templates

By default, the DynamicButtons control will generate the appropriate buttons, separated by " ". Their labels and images come from various properties on the control such as **NewButtonText** and **NewButtonImageUrl**. The predefined output depends on the mode:

For ReadOnly mode with a list-style DataBound control

```
[Edit] [Delete] [Select] [Details]
```

For ReadOnly mode with a single record-style DataBound control

```
[Delete] [Edit] [New]
```

For Edit and Insert modes

```
[Save] [Cancel]
```

You can customize these layouts within these four Layout properties: **LayoutReadOnlyList**, **LayoutReadOnlyRecord**, **LayoutEditMode**, and **LayoutInsertMode**. They use tokens where buttons go, and have powerful rules to show and hide HTML in case the DynamicButtons control has to hide a button. See "Using the Layout Properties" below.

If you need more control of the formatting, or you want to introduce additional web controls, define the complete HTML and web control definition as ASP.NET markup within these three properties: **ReadOnlyTemplate**, **EditTemplate**, and **InsertTemplate**. See "Using Templates" below.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the Layout Properties

Unless you use a Template property, this control's output is generated for you. It includes buttons from DES Button, LinkButton, or ImageButton controls as determined by the **ButtonType** property. It also includes HTML that appears before and after buttons. Use these four properties to define the HTML and positions of controls, as expressed by tokens.

Property Name	Default value
LayoutReadOnlyList	`{EDIT:: }{DELETE:: }{SELECT:: }{DETAILS}`
LayoutReadOnlyRecord	`{DELETE:: }{EDIT:: }{NEW}`
LayoutEditMode	`{SAVE} {CANCEL}`
LayoutInsertMode	`{SAVE} {CANCEL}`

Looking at the default values above, you can see the tokens. They are command names enclosed in brackets, like "{SAVE}". The command names are uppercase. Each mode only supports a specific list of command names. The default values reflect that list. (So if you add "{EDIT}" to the **LayoutEditMode**, it will not create an Edit button.)

One key task of the DynamicButtons control is to hide buttons that are not supported in ReadOnly mode. For example, the Delete button is not shown unless the DataSource.EnableDelete property is true. The Layouts need a way to remove HTML appearing around a button that is hidden. For example, "{DELETE}" is followed by " ". They both should hide together. To solve this, tokens support an extended syntax.

Syntax	Usage
`{COMMANDNAME}`	Create a button. ReadOnly command names are: NEW, EDIT, DELETE, SELECT, and DETAILS. Edit and Insert mode command names are: SAVE and CANCEL.
`{COMMANDNAME:before}`	Create a button. Place the text from "before" prior to the button. If the button is hidden, so is the "before" text.
`{COMMANDNAME::after}`	Create a button. Place the text from "after" after the button. If the button is hidden, so is the "after" text.
`{COMMANDNAME:before:after}`	Create a button. Place the text "before" prior and "after" after the button. All are hidden together.
`{+COMMANDNAME:text}`	Show "text" only if a button with the command name is shown.
`{-COMMANDNAME:text}`	Show "text" only if a button with the command name is hidden. This is often used to replace a button's position with text.

In all of the above cases, anything after the colon character cannot have a literal colon, left bracket or right bracket. If these characters are needed, use their HTML characterset value: `&#58`, `&#123`, `&#125`.

Example

This DynamicButtons control uses Buttons and has a different label for its Save on new record button. It also changes the Layouts to place the buttons in a `<td>` tag.

```
<desDD:DynamicButtons ID="ReadOnlyButtons" runat="server" ButtonType="Button"
   SaveEditButtonText="Add"
   LayoutReadOnlyRecord=
      "<td>{DELETE}{-DELETE: }</td><td>{EDIT}{INSERT: }</td>"
   LayoutReadOnlyList="<td>{EDIT:: }{DELETE:: }{SELECT:: }{DETAILS}</td>"
   LayoutEditMode="<td>{SAVE} {CANCEL}</td>"
   LayoutInsertMode="<td>{SAVE} {CANCEL}</td>" />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using Templates

While the Layout properties (above) give you flexibility, it may not be enough. Use the Template properties to have total control and introduce other web controls. There are separate template properties for each data entry mode: **ReadOnlyTemplate**, **EditTemplate**, and **InsertTemplate**. Once a Template is defined, that mode no longer uses the Layout properties, **ButtonType** property, or any that define the label or image of the buttons.

Always use DES's Button, LinkButton or ImageButton controls.

Here is an example with Templates. They are enclosed in a `<td>` tag. It includes a the VisibilityByCommand Control, which plays an important role here. See "The VisibilityByCommand Control".

```
<desDD:DynamicButtons ID="DynamicButtons1" runat="server" >
<ReadOnlyTemplate>
<td>
   <desDD:VisibilityByCommand ID="EditButtonSeparator" runat="server"
      CommandName="Edit" WhenCommandIsShown="true" >
        <des:Button ID="EditButton" runat="server" Text="Edit"
           CommandName="Edit" CausesValidation="false" />
         
   </desDD:VisibilityByCommand>
   <desDD:VisibilityByCommand ID="DeleteButtonReplacement" runat="server"
      CommandName="Delete" WhenCommandIsShown="true" >
        <des:Button ID="DeleteButton" runat="server" Text="Delete"
            CommandName="Delete" CausesValidation="false" />
          
   </desDD:VisibilityByCommand>
   <desDD:VisibilityByCommand ID="SelectButtonReplacement" runat="server"
      CommandName="Select" WhenCommandIsShown="true" >
        <des:Button ID="SelectButton" runat="server" Text="Select"
           CommandName="Select" CausesValidation="false" />
         
   </desDD:VisibilityByCommand>
   <des:Button ID="DetailsButton" runat="server" Text="Details"
      CommandName="Details" CausesValidation="false" />
</td>
</ReadOnlyTemplate>

<EditTemplate>
<td>
   <des:Button ID="SaveButtonEdit" runat="server" Text="Save" CommandName="Update" />
 
   <des:Button ID="CancelButtonEdit" runat="server" Text="Cancel" CommandName="Cancel"
      CausesValidation="false" />
</td>
</EditTemplate>

<InsertTemplate>
<td>
   <des:Button ID="SaveButtonInsert" runat="server" Text="Save" CommandName="Insert" />
 
   <des:Button ID="CancelButtonInsert" runat="server" Text="Cancel" CommandName="Cancel"
      CausesValidation="false" />
</td>
</InsertTemplate>
</desDD:DynamicButtons>
```

While you have complete control over the output, the DynamicButtons controls still modifies your Button controls, by assigning the correct Validation Group name and values from the Customizer Control (unless you set **ApplyCustomizerRules** to `false`.)

It also will hide buttons based on settings from the DataSource and DataBound Control Adapter objects. If you don't want it to hide them, set the appropriate property to `false`: **AutoHideNewButton**, **AutoHideEditButton**, **AutoHideDeleteButton**, and **AutoHideDetailsButton**. The Select button is hidden if **ShowSelectButton** is `false`.

The act of hiding buttons can leave markup hanging around where it was intended to work with the button present. The VisibilityByCommand control is there to assist you. See "The VisibilityByCommand Control".

For more assistance in designing your buttons interface, see "Using actual Button controls".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Converting from Layout to Template quickly

When you switch to a Template property, it is an empty workspace, lacking any HTML or button definitions. Instead, you may want to start with the HTML and buttons that would be generated without the template. The DynamicButtons control provides a way to create this initial markup.

1. Open the web form or user control in Visual Studio
2. Switch to design mode
3. Point to the DynamicButtons control and click on its SmartTag (▶).
4. Select\the desired mode to edit from the **Mode to view in the designer** field.
5. Choose the **Convert to *Mode*Template** command.

Understanding command names on buttons

Each button uses its **CommandName** property to instruct the DataBound control on the desired action. Here is a summary of the command names for this property:

CommandName	Usage
"New"	Start creating a record.
"Edit"	Start editing a record.
"Delete"	Delete a record. If you set the **AutoCustomizerRules** property to `false`, your Delete button will still be assigned a default confirmation message as its too important to confirm with the user prior to deleting the record. If you want a different confirmation message, assign it to the button's **ConfirmMessage** property.
"Details"	View the record in a single-record interface.
"Select"	Mark the record's row in a list as selected.
"Update"	Save an existing record. Usually this switches out of Edit mode.
"Insert"	Save a new record. Usually this switches out of Insert mode.
"Cancel"	Cancel edits on a new or existing record. Switch out of Edit or Insert mode.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Creating Templates programmatically

It can be tricky creating a Template programmatically. Microsoft recommends that you create a new class that implements ITemplate. When you create the InstantiateIn() method, use it to add the desired controls to the control passed to it. Then programmatically assign an instance of your new class to the desired Template property on the DynamicButtons control.

That solution is somewhat complicated. As an easier alternative, use the **ButtonsCreated** event on the DynamicButtons control. This event fires after buttons are created, whether generated by the Layout properties or Template properties. You are passed the DynamicButtons control. Its child controls are what was generated. Use your **ButtonsCreated** event method to modify the child controls collection. You can eliminate whatever was already generated if you like by calling Controls.Clear() first.

The event takes two parameters. The first is actually the DynamicButtons control, although the event requires it to be a generic object class. So typecast to `PeterBlum.DES.DynamicData.DynamicButtons`.

Here is an example. It clears the existing child controls and adds Save and Cancel buttons separated by " " in Insert mode.

[C#]

```
protected void DynamicButtons1_ButtonsCreated(object sender, EventArgs e)
{
  PeterBlum.DES.DynamicData.DynamicButtons vDynamicButtons =
    (PeterBlum.DES.DynamicData.DynamicButtons) sender;
  if (vDynamicButtons.DataBoundControlAdapter.GetMode() !=
    DataBoundControlMode.Insert)
    return;

  vDynamicButtons.Controls.Clear();
  PeterBlum.DES.LinkButton vSaveButton = new PeterBlum.DES.LinkButton();
  vSaveButton.ID = "SaveNewButton";
  vSaveButton.Text = vDynamicButtons.SaveNewButtonText;
  vSaveButton.CommandName = "Insert";
  vDynamicButtons.Controls.Add(vSaveButton);

  vDynamicButtons.Controls.Add(new LiteralControl(" "));

  PeterBlum.DES.LinkButton vCancelButton = new PeterBlum.DES.LinkButton();
  vCancelButton.ID = "CancelNewButton";
  vCancelButton.Text = vDynamicButtons.CancelNewButtonText;
  vCancelButton.CommandName = "Cancel";
  vCancelButton.CausesValidation = false;
  vDynamicButtons.Controls.Add(vCancelButton);
}
```

VB EXAMPLE ON THE NEXT PAGE

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

[VB]

```
Protected Sub DynamicButtons1_ButtonsCreated(ByVal sender As Object,
  ByVal e As EventArgs)
  Dim vDynamicButtons As PeterBlum.DES.DynamicData.DynamicButtons = _
    CType(sender, PeterBlum.DES.DynamicData.DynamicButtons)
  If vDynamicButtons.DataBoundControlAdapter.GetMode() <>
    DataBoundControlMode.Insert) Then
    Return
  End If

  vDynamicButtons.Controls.Clear()
  Dim vSaveButton As PeterBlum.DES.LinkButton = New PeterBlum.DES.LinkButton()
  vSaveButton.ID = "SaveNewButton"
  vSaveButton.Text = vDynamicButtons.SaveNewButtonText
  vSaveButton.CommandName = "Insert"
  vDynamicButtons.Controls.Add(vSaveButton)

  vDynamicButtons.Controls.Add(New LiteralControl(" "))

  Dim vCancelButton As PeterBlum.DES.LinkButton = New PeterBlum.DES.LinkButton()
  vCancelButton.ID = "CancelNewButton"
  vCancelButton.Text = vDynamicButtons.CancelNewButtonText
  vCancelButton.CommandName = "Cancel"
  vCancelButton.CausesValidation = False
  vDynamicButtons.Controls.Add(vCancelButton)
End Sub
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Adding a DynamicButtons Control

These steps ask you to jump around the document using clicks on links. Adobe Reader offers a **Previous View** command to return to the link. Look for this in the Adobe Reader (shown v6.0)

1. Prepare the page for DES controls. See "Preparing a page for DES controls" in the **General Features Guide**. It covers issues like style sheets, AJAX, and localization.
2. Add an DynamicButtons control to the page. It must go in a Template area of a DataBound control, such as ItemTemplate on ListView, a TemplateField object on GridView, or within a Pattern Template file.

Visual Studio and Visual Web Developer Design Mode Users

Drag the DynamicButtons control from the Toolbox onto your web form.

Text Entry Users

Add the control (inside the <form> area):

```
<desDD:DynamicButtons id="[YourControlID]" runat="server" />
```

Programmatically creating the DynamicButtons control

- Identify the control which you will add the DynamicButtons control to its **Controls** collection. Like all ASP.NET controls, the DynamicButtons can be added to any control that supports child controls, including Templates.
- Create an instance of the DynamicButtons control class. The constructor takes no parameters.
- Assign the **ID** property.
- Add the DynamicButtons control to the **Controls** collection.

In this example, the DynamicButtons is created with an **ID** of "DynamicButtons1". It is added to PlaceHolder1.

[C#]

```
PeterBlum.DES.DynamicData.DynamicButtons vDynamicButtons =
   new PeterBlum.DES.DynamicData.DynamicButtons();
vDynamicButtons.ID = "DynamicButtons1";
PlaceHolder1.Controls.Add(vDynamicButtons);
```

*Note: The namespace for these controls is PeterBlum.DES. If you prefer, add a **using** clause to that namespace on your form.*

[VB]

```
Dim vDynamicButtons As PeterBlum.DES.DynamicData.DynamicButtons = _
   New PeterBlum.DES.DynamicData.DynamicButtons()
vDynamicButtons.ID = "DynamicButtons1"
PlaceHolder1.Controls.Add(vDynamicButtons)
```

*Note: The namespace for these controls is PeterBlum.DES.DynamicData. If you prefer, add an **Imports** clause to that namespace on your form.*

CONTINUED ON THE NEXT PAGE

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

3. Set the properties associated with the DynamicButtons. See "Properties for the DynamicButtons Control".

   When you do not use Templates, set the **ButtonType** to the desired type: `Button`, `Link`, or `Image`. If you choose `Image`, be sure that each ***buttonname*ImageUrl** property has a path to the image file. If you need to change a button's label, use the ***buttonname*Text** property. See "Using the Layout Properties".

   When you use Templates, see "Using Templates".

Guidelines for setting properties

- Design mode users can use the Properties Editor or the Expanded Properties Editor. (See "Expanded Properties Editor" in the **General Features Guide**.) The SmartTag also offers some of the most important properties.
- Text entry users should add the properties into the `<desDD:ControlClass>` tag in this format: `propertyname="value"`
- When setting a property programmatically, have a reference to the control's object and set the property according to your language's rules.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties for the DynamicButtons Control

The PeterBlum.DES.DynamicData.DynamicButtons control is subclassed from `System.Web.UI.WebControls.WebControl`.

Click on any of these topics to jump to them:

- Button Visibility Rules properties
- Buttons without Templates properties
  - Button Appearance properties
  - Button Layout properties
  - New Record Button properties
  - Edit Record Button properties
  - Delete Record Button properties
  - Details Button properties
  - Select a Row Button properties
  - Save New Record Button properties
  - Cancel New Record Button properties
  - Save Edit Record Button properties
  - Cancel Edit Record Button properties
- Button Templates properties
- Container Appearance properties
- Behavior Properties
- ButtonsCreating Event Handler
- ButtonsCreated Event Handler

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Button Visibility Rules properties

These properties are found in the "Button Visibility Rules" category of the Visual Studio Properties Editor.

- **ShowSelectButton** (Boolean) – Determines if the Select button is shown or not. It is applied both to Template and non-template setups. By default it is shown. However, if the **LayoutReadOnlyList** property omits the "{SELECT}" token or **ReadOnlyTemplate** lacks an actual Button control for the Select command, it will never be shown.

  It defaults to `true`.

- **AutoHideEditButton** (Boolean) – Determines if the Edit button should be hidden based on these rules:
  - The DataBound control is in ReadOnly mode
  - The **IDataSource.EnableUpdate** property is `false` and the DataBound Control Adapter's **SupportsEditMode** property is `false`.
  - The button's **CommandName** property is `"Edit"`.

  Note: The Edit button edits using the current DataBound control when IDataSource.EnableUpdate is true. If EnableUpdate is false and SupportsEditMode is true, it uses a different DataBound control to edit, such as redirecting to another Page Template or when using the ListDetails.aspx Page Template, using the DynamicFormView control on the same page.

  When `true`, it will be hidden if all of the above rules are met.

  When `false`, it will not be automatically hidden. You can still hide it within the ItemCreated or RowCreated event of the DataBound control.

  It defaults to `true`.

- **AutoHideNewButton** (Boolean) – Determines if the New record button should be hidden based on these rules:
  - The DataBound control is in ReadOnly mode
  - The **IDataSource.EnableInsert** property is `false` and the DataBound Control Adapter's **SupportsInsertMode** property is `false`.
  - The button's **CommandName** property is `"New"`.

  Note: The Insert button inserts using the current DataBound control when IDataSource.EnableInsert is true. If EnableUpdate is false and SupportsInsertMode is true, it uses a different DataBound control to edit, such as redirecting to another Page Template or when using the ListDetails.aspx Page Template, using the DynamicFormView control on the same page.

  When `true`, it will be hidden if all of the above rules are met.

  When `false`, it will not be automatically hidden. You can still hide it within the ItemCreated or RowCreated event of the DataBound control.

  It defaults to `true`.

- **AutoHideDeleteButton** (Boolean) – Determines if the Delete button should be hidden based on these rules:
  - The DataBound control is in ReadOnly mode
  - The **IDataSource.EnableDelete** property is `false`.
  - The button's **CommandName** property is `"Delete"`.

  Note: The Delete button edits using the current DataBound control when IDataSource.EnableDelete is true.

  When `true`, it will be hidden if all of the above rules are met.

  When `false`, it will not be automatically hidden. You can still hide it within the ItemCreated or RowCreated event of the DataBound control.

  It defaults to `true`.

- **AutoHideDetailsButton** (Boolean) – Determines if the Details button should be hidden based on these rules:

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- The DataBound control is in ReadOnly mode
- The DataBound control is a single record type OR

  the DataBound Control Adapter's **ConnectedToControlID** property is assigned OR

  the command will not redirect to another page.

When `true`, it will be hidden if all of the above rules are met.

When `false`, it will not be automatically hidden. You can still hide it within the ItemCreated or RowCreated event of the DataBound control.

It defaults to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Buttons without Templates properties

These properties only apply when there is no Template property setup for the data entry mode. They are built around the Layout properties. See "Using the Layout Properties".

These properties are found in the "Buttons without Templates" category of the Visual Studio Properties Editor.

Click on any of these topics to jump to them:

- Button Appearance properties
- Button Layout properties
- New Record Button properties
- Edit Record Button properties
- Delete Record Button properties
- Details Button properties
- Select a Row Button properties
- Save New Record Button properties
- Cancel New Record Button properties
- Save Edit Record Button properties
- Cancel Edit Record Button properties

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Button Appearance properties

- **ButtonType** (enum System.Web.UI.WebControls.ButtonType) – The type of button. Choose from `Button`, `Link` (which creates a LinkButton), or `Image` (which creates an Image button).

  When set to `Image`, be sure to assign the ***buttonname*ImageUrl** properties to specify the images. If not assigned, the button will switch back to a Link button.

  When set to `Link`, it can use both text and images. You must have the ***buttonname*ImageUrl** property specifying an image. Use the **LinkButtonLayout** and **LinkButtonLayoutSeparator** properties to define how text and image are placed on the page.

  It defaults to `ButtonType.Link`.

- **ButtonCssClass** (string) – The style sheet class applied to each button. It defaults to "".

- **LinkButtonLayout** (enum PeterBlum.DES.DynamicData.LinkButtonLayout) – When using **ButtonType**=`Link`, it can show either or both text and image. This determines the actual layout. When set to `Auto`, only one of the two are shown. Otherwise both are shown.

  The enumerated type `PeterBlum.DES.DynamicData.LinkButtonLayout` has these values:

  - `Auto` - Only one is shown. If the ImageUrl is defined, an image is shown. Otherwise the text is shown.
  - `ImageOutside` - Both are shown. On the First and Previous button, the image is to the left of the text. On the Next and Last button, the image is to the right of the text. This option uses **LinkButtonLayoutSeparator** between the two elements.
  - `ImageInside` - Both are shown. On the First and Previous button, the image is to the right of the text. On the Next and Last button, the image is to the left of the text. This option uses **LinkButtonLayoutSeparator** between the two elements.

  It defaults to `LinkButtonLayout.Auto`.

- **LinkButtonLayoutSeparator** (string) – HTML placed in between an image and textual LinkButton when **ButtonType**=`Link` and the **LinkButtonLayout** property is set to `ImageInside` or `ImageOutside`.

  It defaults to "` `".

- **MultipleImages** (boolean) – When `true` and buttons are using images, you have images for pressed and mouseover effects. The ***buttonname*ImageUrl** property points to the normal graphic. Pressed and MouseOver are files in the same folder with the "Pressed" and "MouseOver" inserted in name, just before the file extension.

  The code will detect the presence of these files. If neither are found, it is not used. If one is found, it is used and the other is not. So it is safe to set this to true all of the time, except for the extra overhead of time used.

  The **PeterBlum.DES.Globals.Page.EnableButtonImageEffects** property can override this behavior.

  When `false`, it does not look for multiple images. However, the ***buttonname*ImageUrl** can override this by having a pipe delimited list of URLs in this format: `Normal|Pressed|MouseOver`.

  When `true`, it looks for multiple images.

  It defaults to `false`.

  License Note: This property requires a license for Peter's Interactive Pages.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Button Layout properties

- **LayoutReadOnlyList** (string) – Determines the HTML that lays out this control on a list-style DataBound control that is in ReadOnly mode. See "Using the Layout Properties".

  Use these tokens to represent where the various buttons go: {EDIT}, {DELETE}, {DETAILS}, and {SELECT}.

  The default is `"{EDIT:: }{DELETE:: }{SELECT:: }{DETAILS}"`.

  The following constants are available on the DynamicButtons class. Use them to set **LayoutReadOnlyList** either programmatically or by using DataBinding notation in the markup.

cLayoutReadOnlyList	`"{EDIT:: }{DELETE:: }{SELECT:: }{DETAILS}"`

cLayoutReadOnlyListInTableCell	`"<td>`
	`{EDIT:: }{DELETE:: }{SELECT:: }{DETAILS}`
	`</td>"`

- **LayoutReadOnlyRecord** (string) – Determines the HTML that lays out this control on a single record-style DataBound control that is in ReadOnly mode. See "Using the Layout Properties".

  Use these tokens to represent where the various buttons go: {NEW}, {EDIT}, and {DELETE}.

  The default is `"{DELETE:: }{EDIT:: }{NEW}"`.

  The following constants are available on the DynamicButtons class. Use them to set **LayoutReadOnlyRecord** either programmatically or by using DataBinding notation in the markup.

cLayoutReadOnlyRecord	`"{DELETE:: }{EDIT:: }{NEW}"`
cLayoutReadOnlyRecordInTableCell	`"<td>{DELETE:: }{EDIT:: }{NEW}</td>"`

cLayoutReadOnlyRecordInTwoTableCells	`"<td>`
	`{DELETE}{-DELETE: }</td><td>{EDIT}{INSERT: }`
	`</td>"`

- **LayoutEditMode** (string) – Determines the HTML that lays out this control on a DataBound control that is in Edit mode. See "Using the Layout Properties".

  Use these tokens to represent where the various buttons go: {SAVE}, and {CANCEL}.

  The default is `"{SAVE} {CANCEL}"`.

  The following constants are available on the DynamicButtons class. Use them to set **LayoutEditMode** either programmatically or by using DataBinding notation in the markup.

cLayoutEditMode	`"{SAVE} {CANCEL}"`
cLayoutEditModeInTableCell	`"<td>{SAVE} {CANCEL}</td>"`

- **LayoutInsertMode** (string) – Determines the HTML that lays out this control on a DataBound control that is in Insert mode. See "Using the Layout Properties".

  Use these tokens to represent where the various buttons go: {SAVE}, and {CANCEL}.

  The default is `"{SAVE} {CANCEL}"`.

  The following constants are available on the DynamicButtons class. Use them to set **LayoutEditMode** either programmatically or by using DataBinding notation in the markup.

cLayoutEditMode	`"{SAVE} {CANCEL}"`
cLayoutEditModeInTableCell	`"<td>{SAVE} {CANCEL}</td>"`

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

New Record Button properties

- **NewButtonText** (string) – Determines the text for the New Record button. When using ImageButtons, this is the alternative text for a missing image. It defaults to "`New`".

- **NewButtonTextLookupID** (string) – Gets the value for **NewButtonText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **NewButtonText** 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 "".

- **NewButtonImageUrl** (string) – The URL to an image that is shown on the New Record button. Does not apply when **ButtonType**=Button. When using **ButtonType**=Image and this is unassigned, a LinkButton is shown.

  It defaults to "".

  #### Special Symbols for URLs

  The "{APPEARANCE}" token will be replaced by the default path to the **Appearance** folder, which you defined as you set up the web site.

  Supports the use of the tilde (~) as the first character to be replaced by the virtual path to the web application.

  #### Images for Pressed and MouseOver Effects

  You can have images for pressed and mouseover effects as well as the normal image. The names of the image files determine their purpose. Define the name of the normal image. For example, "myimage.gif". Create the pressed version and give it the same name, with "Pressed" added before the extension. For example, "myimage*pressed*.gif". Create the mouseover version and give it the same name, with "MouseOver" added before the extension. For example, myimage*mouseover*.gif.

  The **NewButtonImageUrl** 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 and setting **MultipleImages** to `false`. 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`.

- **NewButtonWhenDisabledImageUrl** (string) – The image for the New Record command button when disabled. The URL to an image that is shown when the ButtonType is Image or Link.

  If not assigned, the **NewButtonImageUrl** is used.

  It defaults to "".

  #### 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.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Edit Record Button properties

- **EditButtonText** (string) – Determines the text for the Edit Record button. When using ImageButtons, this is the alternative text for a missing image. It defaults to "`Edit`".

- **EditButtonTextLookupID** (string) – Gets the value for **EditButtonText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **EditButtonText** 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 "".

- **EditButtonImageUrl** (string) – The URL to an image that is shown on the Edit Record button. Does not apply when **ButtonType**=Button. When using **ButtonType**=Image and this is unassigned, a LinkButton is shown.

  It defaults to "".

  #### Special Symbols for URLs

  The "{APPEARANCE}" token will be replaced by the default path to the **Appearance** folder, which you defined as you set up the web site.

  Supports the use of the tilde (~) as the first character to be replaced by the virtual path to the web application.

  #### Images for Pressed and MouseOver Effects

  You can have images for pressed and mouseover effects as well as the normal image. The names of the image files determine their purpose. Define the name of the normal image. For example, "myimage.gif". Create the pressed version and give it the same name, with "Pressed" added before the extension. For example, "myimage*pressed*.gif". Create the mouseover version and give it the same name, with "MouseOver" added before the extension. For example, myimage*mouseover*.gif.

  The **EditButtonImageUrl** 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 and setting **MultipleImages** to `false`. 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`.

- **EditButtonWhenDisabledImageUrl** (string) – The image for the Edit Record command button when disabled. The URL to an image that is shown when the ButtonType is Image or Link.

  If not assigned, the **EditButtonImageUrl** is used.

  It defaults to "".

  #### 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.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Delete Record Button properties

- **DeleteButtonText** (string) – Determines the text for the Delete Record button. When using ImageButtons, this is the alternative text for a missing image. It defaults to "`Delete`".

- **DeleteButtonTextLookupID** (string) – Gets the value for **DeleteButtonText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **DeleteButtonText** 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 "".

- **DeleteButtonImageUrl** (string) – The URL to an image that is shown on the Delete Record button. Does not apply when **ButtonType**=Button. When using **ButtonType**=Image and this is unassigned, a LinkButton is shown.

  It defaults to "".

  #### Special Symbols for URLs

  The "{APPEARANCE}" token will be replaced by the default path to the **Appearance** folder, which you defined as you set up the web site.

  Supports the use of the tilde (~) as the first character to be replaced by the virtual path to the web application.

  #### Images for Pressed and MouseOver Effects

  You can have images for pressed and mouseover effects as well as the normal image. The names of the image files determine their purpose. Define the name of the normal image. For example, "myimage.gif". Create the pressed version and give it the same name, with "Pressed" added before the extension. For example, "myimage*pressed*.gif". Create the mouseover version and give it the same name, with "MouseOver" added before the extension. For example, myimage*mouseover*.gif.

  The **DeleteButtonImageUrl** 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 and setting **MultipleImages** to `false`. 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`.

- **DeleteButtonWhenDisabledImageUrl** (string) – The image for the Delete Record command button when disabled. The URL to an image that is shown when the ButtonType is Image or Link.

  If not assigned, the **DeleteButtonImageUrl** is used.

  It defaults to "".

  #### 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.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Details Button properties

- **DetailsButtonText** (string) – Determines the text for the Details button. When using ImageButtons, this is the alternative text for a missing image. It defaults to "`Details`".

- **DetailsButtonTextLookupID** (string) – Gets the value for **DetailsButtonText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **DetailsButtonText** 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 "".

- **DetailsButtonImageUrl** (string) – The URL to an image that is shown on the Details button. Does not apply when **ButtonType**=Button. When using **ButtonType**=Image and this is unassigned, a LinkButton is shown.

  It defaults to "".

  #### Special Symbols for URLs

  The "{APPEARANCE}" token will be replaced by the default path to the **Appearance** folder, which you defined as you set up the web site.

  Supports the use of the tilde (~) as the first character to be replaced by the virtual path to the web application.

  #### Images for Pressed and MouseOver Effects

  You can have images for pressed and mouseover effects as well as the normal image. The names of the image files determine their purpose. Define the name of the normal image. For example, "myimage.gif". Create the pressed version and give it the same name, with "Pressed" added before the extension. For example, "myimage*pressed*.gif". Create the mouseover version and give it the same name, with "MouseOver" added before the extension. For example, myimage*mouseover*.gif.

  The **DetailsButtonImageUrl** 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 and setting **MultipleImages** to `false`. 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`.

- **DetailsButtonWhenDisabledImageUrl** (string) – The image for the Details command button when disabled. The URL to an image that is shown when the ButtonType is Image or Link.

  If not assigned, the **DetailsButtonImageUrl** is used.

  It defaults to "".

  #### 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.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Select a Row Button properties

- **SelectButtonText** (string) – Determines the text for the Select a Row button. When using ImageButtons, this is the alternative text for a missing image. It defaults to "`Select`".

- **SelectButtonTextLookupID** (string) – Gets the value for **SelectButtonText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **SelectButtonText** 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 "".

- **SelectButtonImageUrl** (string) – The URL to an image that is shown on the Select a Row button. Does not apply when **ButtonType**=Button. When using **ButtonType**=Image and this is unassigned, a LinkButton is shown.

  It defaults to "".

  #### Special Symbols for URLs

  The "{APPEARANCE}" token will be replaced by the default path to the **Appearance** folder, which you defined as you set up the web site.

  Supports the use of the tilde (~) as the first character to be replaced by the virtual path to the web application.

  #### Images for Pressed and MouseOver Effects

  You can have images for pressed and mouseover effects as well as the normal image. The names of the image files determine their purpose. Define the name of the normal image. For example, "myimage.gif". Create the pressed version and give it the same name, with "Pressed" added before the extension. For example, "myimage*pressed*.gif". Create the mouseover version and give it the same name, with "MouseOver" added before the extension. For example, myimage*mouseover*.gif.

  The **SelectButtonImageUrl** 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 and setting **MultipleImages** to `false`. 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`.

- **SelectButtonWhenDisabledImageUrl** (string) – The image for the Select a Row command button when disabled. The URL to an image that is shown when the ButtonType is Image or Link.

  If not assigned, the **SelectButtonImageUrl** is used.

  It defaults to "".

  #### 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.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Save New Record Button properties

- **SaveNewButtonText** (string) – Determines the text for the Save New Record button. When using ImageButtons, this is the alternative text for a missing image. It defaults to "`Save`".

- **SaveNewButtonTextLookupID** (string) – Gets the value for **SaveNewButtonText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **SaveNewButtonText** 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 "".

- **SaveNewButtonImageUrl** (string) – The URL to an image that is shown on the Save New Record button. Does not apply when **ButtonType**=Button. When using **ButtonType**=Image and this is unassigned, a LinkButton is shown.

  It defaults to "".

  #### Special Symbols for URLs

  The "{APPEARANCE}" token will be replaced by the default path to the **Appearance** folder, which you defined as you set up the web site.

  Supports the use of the tilde (~) as the first character to be replaced by the virtual path to the web application.

  #### Images for Pressed and MouseOver Effects

  You can have images for pressed and mouseover effects as well as the normal image. The names of the image files determine their purpose. Define the name of the normal image. For example, "myimage.gif". Create the pressed version and give it the same name, with "Pressed" added before the extension. For example, "myimage*pressed*.gif". Create the mouseover version and give it the same name, with "MouseOver" added before the extension. For example, myimage*mouseover*.gif.

  The **SaveNewButtonImageUrl** 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 and setting **MultipleImages** to `false`. 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`.

- **SaveNewButtonWhenDisabledImageUrl** (string) – The image for the Save New Record command button when disabled. The URL to an image that is shown when the ButtonType is Image or Link.

  If not assigned, the **SaveNewButtonImageUrl** is used.

  It defaults to "".

  #### 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.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Cancel New Record Button properties

- **CancelNewButtonText** (string) – Determines the text for the Cancel New Record button. When using ImageButtons, this is the alternative text for a missing image. It defaults to "`Cancel`".

- **CancelNewButtonTextLookupID** (string) – Gets the value for **CancelNewButtonText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **CancelNewButtonText** 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 "".

- **CancelNewButtonImageUrl** (string) – The URL to an image that is shown on the Cancel New Record button. Does not apply when **ButtonType**=Button. When using **ButtonType**=Image and this is unassigned, a LinkButton is shown.

  It defaults to "".

  #### Special Symbols for URLs

  The "{APPEARANCE}" token will be replaced by the default path to the **Appearance** folder, which you defined as you set up the web site.

  Supports the use of the tilde (~) as the first character to be replaced by the virtual path to the web application.

  #### Images for Pressed and MouseOver Effects

  You can have images for pressed and mouseover effects as well as the normal image. The names of the image files determine their purpose. Define the name of the normal image. For example, "myimage.gif". Create the pressed version and give it the same name, with "Pressed" added before the extension. For example, "myimage*pressed*.gif". Create the mouseover version and give it the same name, with "MouseOver" added before the extension. For example, myimage*mouseover*.gif.

  The **CancelNewButtonImageUrl** 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 and setting **MultipleImages** to `false`. 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`.

- **CancelNewButtonWhenDisabledImageUrl** (string) – The image for the Cancel New Record button when disabled. The URL to an image that is shown when the ButtonType is Image or Link.

  If not assigned, the **CancelNewButtonImageUrl** is used.

  It defaults to "".

  #### 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.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Save Edit Record Button properties

- **SaveEditButtonText** (string) – Determines the text for the Save Edit Record button. When using ImageButtons, this is the alternative text for a missing image. It defaults to "`Save`".

- **SaveEditButtonTextLookupID** (string) – Gets the value for **SaveEditButtonText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **SaveEditButtonText** 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 "".

- **SaveEditButtonImageUrl** (string) – The URL to an image that is shown on the Save Edit Record button. Does not apply when **ButtonType**=Button. When using **ButtonType**=Image and this is unassigned, a LinkButton is shown.

  It defaults to "".

  #### Special Symbols for URLs

  The "{APPEARANCE}" token will be replaced by the default path to the **Appearance** folder, which you defined as you set up the web site.

  Supports the use of the tilde (~) as the first character to be replaced by the virtual path to the web application.

  #### Images for Pressed and MouseOver Effects

  You can have images for pressed and mouseover effects as well as the normal image. The names of the image files determine their purpose. Define the name of the normal image. For example, "myimage.gif". Create the pressed version and give it the same name, with "Pressed" added before the extension. For example, "myimage*pressed*.gif". Create the mouseover version and give it the same name, with "MouseOver" added before the extension. For example, myimage*mouseover*.gif.

  The **SaveEditButtonImageUrl** 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 and setting **MultipleImages** to `false`. 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`.

- **SaveEditButtonWhenDisabledImageUrl** (string) – The image for the Save Edit Record button when disabled. The URL to an image that is shown when the ButtonType is Image or Link.

  If not assigned, the **SaveEditButtonImageUrl** is used.

  It defaults to "".

  #### 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.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Cancel Edit Record Button properties

- **CancelEditButtonText** (string) – Determines the text for the Cancel Edit Record button. When using ImageButtons, this is the alternative text for a missing image. It defaults to "`New`".

- **CancelEditButtonTextLookupID** (string) – Gets the value for **CancelEditButtonText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **CancelEditButtonText** 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 "".

- **CancelEditButtonImageUrl** (string) – The URL to an image that is shown on the Cancel Edit Record button. Does not apply when **ButtonType**=Button. When using **ButtonType**=Image and this is unassigned, a LinkButton is shown.

  It defaults to "".

  #### Special Symbols for URLs

  The "{APPEARANCE}" token will be replaced by the default path to the **Appearance** folder, which you defined as you set up the web site.

  Supports the use of the tilde (~) as the first character to be replaced by the virtual path to the web application.

  #### Images for Pressed and MouseOver Effects

  You can have images for pressed and mouseover effects as well as the normal image. The names of the image files determine their purpose. Define the name of the normal image. For example, "myimage.gif". Create the pressed version and give it the same name, with "Pressed" added before the extension. For example, "myimage*pressed*.gif". Create the mouseover version and give it the same name, with "MouseOver" added before the extension. For example, myimage*mouseover*.gif.

  The **CancelEditButtonImageUrl** 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 and setting **MultipleImages** to `false`. 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`.

- **CancelEditButtonWhenDisabledImageUrl** (string) – The image for the Cancel Edit Button command button when disabled. The URL to an image that is shown when the ButtonType is Image or Link.

  If not assigned, the **CancelEditButtonImageUrl** is used.

  It defaults to "".

  #### 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.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Button Templates properties

These properties let you create button and HTML output explicity. See “Using Templates”.

These properties are found in the “Button Templates” category of the Visual Studio Properties Editor.

- **ReadOnlyTemplate** (ITemplate) – The buttons, HTML and other web controls used in ReadOnly mode. When this property is assigned, it overrides automatic button generation in ReadOnly mode. For details, see “Using Templates”.
- **EditTemplate** (ITemplate) – The buttons, HTML and other web controls used in Edit mode. When this property is assigned, it overrides automatic button generation in Edit mode. For details, see “Using Templates”.
- **InsertTemplate** (ITemplate) – The buttons, HTML and other web controls used in Insert mode. When this property is assigned, it overrides automatic button generation in Insert mode. For details, see “Using Templates”.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Container Appearance properties

The entire DynamicButtons control's output can be enclosed in a or tag with a style sheet applied. Use these properties to set it up.

These properties are found in the "Container Appearance" category of the Visual Studio Properties Editor.

- **ContainerTag** (enum PeterBlum.DES.DynamicData.DynamicButtonsContainerTag) – Specifies the HTML tag to output around the buttons. Use this when you need a style sheet on the container around the buttons. When set to `None`, the buttons are not enclosed by a tag.

  When it lacks the tag that you want, set it to `None` and create the HTML tag around the DynamicButtons control.

  If you set **CssClass**, it will change this from `None` to `DIV`. If its already `SPAN`, no change occurs. This is because **CssClass** requires some kind of container.

  It defaults to `DynamicButtonsContainerTag.None`.

- **CssClass** (string) – The style sheet class assigned to the HTML container around the buttons. When you set it, it will change **ContainerTag** from `None` to `DIV`. If **ContainerTag** is `None`, **CssClass** property is ignored.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Behavior Properties

The Properties Editor shows these properties in the Behavior category.

- **InAJAXUpdate** (Boolean) – When using AJAX on this page, set this to `true` if the control is involved in an AJAX update. See "Using These Controls with AJAX" in the **General Features Guide**. It defaults to `false`.
- **Visible** (Boolean) – When `false`, this control is not used. It defaults to `true`.
- **ReadOnlyTableBehavior** (enum PeterBlum.DES.DynamicData.ReadOnlyTableBehavior) – When the MetaTable is ReadOnly, this determines the user interface applied to the buttons. They can be shown, disabled, or hidden.

  The enumerated type `PeterBlum.DES.DynamicData.ReadOnlyTableBehavior` has these values:

  - `None` - No change to the current control. Generally that keeps it enabled and visible.
  - `Disable` - Disable the buttons.
  - `Hide` - Make the buttons Visible=`false`.

  It defaults to `ReadOnlyTableBehavior.Hide`.
- **ApplyCustomizerRules** (Boolean) – When `true`, applies the following properties from the Customizer control to the buttons contained within this control: **DisableOnSubmit**, **MayMoveOnClick**, **ChangeMonitorEnables**, **DeleteConfirmMessage**, **UpdateConfirmMessage**, and **CancelConfirmMessage**.

  It does this whether buttons are automatically generated or within a Template.

  It always uses the Customizer Control identified by the DataBound Control Adapter.

  *Note: Even if there is no Customizer control, the **DeleteConfirmMessage** is assigned to a default so long as the associated button does not have its own ConfirmMessage set.*

  It defaults to `true`.
- **SetEnterKeyToSaveButton** (Boolean) – When `true`, the Save button will intercept the ENTER key instead of whatever button the browser assigns as the default button. This only applies when the data entry mode is Edit or Insert.

  The Save button is identified by having the **CausesValidation** property set to `true` or **CommandName** of "Update" or "Insert".

  It defaults to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

ButtonsCreating Event Handler

The **ButtonsCreating** event is fired before the DynamicButtons control has converted its properties into the child controls to be output. Use it to modify the properties of the DynamicButtons control.

The event takes two parameters. The first is actually the DynamicButtons control, although the event requires it to be a generic object class. So typecast to `PeterBlum.DES.DynamicData.DynamicButtons`.

Here is an example. It edits the label on the Save button.

[C#]

```
protected void DynamicButtons1_ButtonsCreating(object sender, EventArgs e)
{
  PeterBlum.DES.DynamicData.DynamicButtons vDynamicButtons =
    (PeterBlum.DES.DynamicData.DynamicButtons) sender;
  vDynamicButtons.SaveNewButtonText = "Insert";
}
```

[VB]

```
Protected Sub DynamicButtons1_ButtonsCreating(ByVal sender As Object,
  ByVal e As EventArgs)
  Dim vDynamicButtons As PeterBlum.DES.DynamicData.DynamicButtons = _
    CType(sender, PeterBlum.DES.DynamicData.DynamicButtons)
  vDynamicButtons.SaveNewButtonText = "Insert"
End Sub
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

ButtonsCreated Event Handler

The **ButtonsCreated** event is fired after the DynamicButtons control has converted its properties into the child controls to be output. Use it to modify the child controls. You may elect to replace them, by first calling DynamicButtons.Control.Clear().

The event takes two parameters. The first is actually the DynamicButtons control, although the event requires it to be a generic object class. So typecast to `PeterBlum.DES.DynamicData.DynamicButtons`.

Here is an example. It clears the existing child controls and adds Save and Cancel buttons separated by " " in Insert mode.

[C#]

```
protected void DynamicButtons1_ButtonsCreated(object sender, EventArgs e)
{
  PeterBlum.DES.DynamicData.DynamicButtons vDynamicButtons =
    (PeterBlum.DES.DynamicData.DynamicButtons) sender;
  if (vDynamicButtons.DataBoundControlAdapter.GetMode() !=
    DataBoundControlMode.Insert)
    return;

  vDynamicButtons.Controls.Clear();
  PeterBlum.DES.LinkButton vSaveButton = new PeterBlum.DES.LinkButton();
  vSaveButton.ID = "SaveNewButton";
  vSaveButton.Text = vDynamicButtons.SaveNewButtonText;
  vSaveButton.CommandName = "Insert";
  vDynamicButtons.Controls.Add(vSaveButton);

  vDynamicButtons.Controls.Add(new LiteralControl(" "));

  PeterBlum.DES.LinkButton vCancelButton = new PeterBlum.DES.LinkButton();
  vCancelButton.ID = "CancelNewButton";
  vCancelButton.Text = vDynamicButtons.CancelNewButtonText;
  vCancelButton.CommandName = "Cancel";
  vCancelButton.CausesValidation = false;
  vDynamicButtons.Controls.Add(vCancelButton);
}
```

VB EXAMPLE ON THE NEXT PAGE

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

[VB]

```
Protected Sub DynamicButtons1_ButtonsCreated(ByVal sender As Object,
  ByVal e As EventArgs)
  Dim vDynamicButtons As PeterBlum.DES.DynamicData.DynamicButtons = _
    CType(sender, PeterBlum.DES.DynamicData.DynamicButtons)
  If vDynamicButtons.DataBoundControlAdapter.GetMode() <>
    DataBoundControlMode.Insert) Then
    Return
  End If

  vDynamicButtons.Controls.Clear()
  Dim vSaveButton As PeterBlum.DES.LinkButton = New PeterBlum.DES.LinkButton()
  vSaveButton.ID = "SaveNewButton"
  vSaveButton.Text = vDynamicButtons.SaveNewButtonText
  vSaveButton.CommandName = "Insert"
  vDynamicButtons.Controls.Add(vSaveButton)

  vDynamicButtons.Controls.Add(New LiteralControl(" "))

  Dim vCancelButton As PeterBlum.DES.LinkButton = New PeterBlum.DES.LinkButton()
  vCancelButton.ID = "CancelNewButton"
  vCancelButton.Text = vDynamicButtons.CancelNewButtonText
  vCancelButton.CommandName = "Cancel"
  vCancelButton.CausesValidation = False
  vDynamicButtons.Controls.Add(vCancelButton)
End Sub
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

The VisibilityByCommand Control

The **VisibilityByCommand** control is used within Templates on the DynamicButtons control. Its purpose is to show or hide elements of the template – Buttons and HTML – based on when the DynamicButtons control hides a command. The idea is that HTML next to a button may need to be hidden if the button is hidden. Use the VisibilityByCommand control to group both the button and its nearby HTML. Set its **CommandName** property to the command name on the button.

Sometimes when a button is hidden, it should be replaced by other HTML or web controls. The VisibilityByCommand control handles this case too. Add the replacement HTML and controls as child controls of the VisibilityByCommand control. Then assign the **CommandName** property to the command name of the button being hidden.

You can see them in use within these Pattern Template files, all which use a DynamicButtons control: **ButtonsForList.ascx**, **LinkButtonsForList.ascx**, **DetailsViewButtons.ascx**, and **DetailsViewLinkButtons.ascx**.

Example: Hiding a button and its HTML

```
<desDD:DynamicButtons ID="DynamicButtons1" runat="server" >
<ReadOnlyTemplate>
<td>
   <desDD:VisibilityByCommand ID="EditButtonSeparator" runat="server"
      CommandName="Edit" WhenCommandIsShown="true" >
        <des:Button ID="EditButton" runat="server" Text="Edit"
           CommandName="Edit" CausesValidation="false" />
         
   </desDD:VisibilityByCommand>
   <desDD:VisibilityByCommand ID="DeleteButtonReplacement" runat="server"
      CommandName="Delete" WhenCommandIsShown="true" >
        <des:Button ID="DeleteButton" runat="server" Text="Delete"
            CommandName="Delete" CausesValidation="false" />
        
   </desDD:VisibilityByCommand>
   <desDD:VisibilityByCommand ID="SelectButtonReplacement" runat="server"
      CommandName="Select" WhenCommandIsShown="true" >
        <des:Button ID="SelectButton" runat="server" Text="Select"
           CommandName="Select" CausesValidation="false" />
         
   </desDD:VisibilityByCommand>
   <des:Button ID="DetailsButton" runat="server" Text="Details"
      CommandName="Details" CausesValidation="false" />
</td>
</ReadOnlyTemplate>
</desDD:DynamicButtons>
```

Example: Showing HTML when a command is hidden

```
<desDD:DynamicButtons ID="DynamicButtons1" runat="server" >
<ReadOnlyTemplate>
<td>
   <des:Button ID="DeleteButton" runat="server" Text="Delete"
      CommandName="Delete" CausesValidation="false" />
   <desDD:VisibilityByCommand ID="DeleteButtonReplacement" runat="server"
      CommandName="Delete" WhenCommandIsShown="false" > </desDD:VisibilityByCommand>
</td>
<td>
   <desDD:VisibilityByCommand ID="EditButtonSeparator" runat="server"
      CommandName="Edit" WhenCommandIsShown="true" >
        <des:Button ID="EditButton" runat="server" Text="Edit"
           CommandName="Edit" CausesValidation="false" />
         
   </desDD:VisibilityByCommand>
   <des:Button ID="NewButton" runat="server" Text="New"
      CommandName="New" CausesValidation="false" />
</td>
</ReadOnlyTemplate>
</desDD:DynamicButtons>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

The DynamicColumnTitle Control

The DynamicColumnTitle Control provides a column title showing a data field's name and optionally allows sorting. It determines if the data field supports sorting and uses a LinkButton + Image instead of the default DynamicLabel. A click on the LinkButton invokes a sort. The image reflects the sort direction and can be clicked to switch the sort direction.

Generally this is used in a list-style Databound control's header row and has been used in the **GridView.ascx** Pattern Template's "ColumnTitles" NamedPart to given the default DynamicListView control a sorting user interface. The actual sorting is handled through features of the Databound control.

Click on any of these topics to jump to them:

- Using the DynamicColumnTitle control
- Properties of the DynamicColumnTitle control


```
<desDD:DynamicColumnTitle ID="DynamicColumnTitle1" runat="server"  />
<desDD:DynamicColumnTitle ID="DynamicColumnTitle2" runat="server"
      TextSource="ShortDisplayName" />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the DynamicColumnTitle control

The DynamicColumnTitle control is subclassed from the DynamicLabel control. When not sorting, this is just an ordinary DynamicLabel. Use its **CssClass** property to apply styles. When sorting is established for the column, the following applies.

Click on any of these topics to jump to them:

- Establishing Sorting on a Data Field
- Establishing the DataField name
- Choosing a source for the column's metadata
- Formatting the text
- Sorting Interface Appearance

Establishing Sorting on a Data Field

The DynamicColumnTitle replaces the label interface with a LinkButton and Image when it detects the data field supports sorting. The following characteristics determine if the data field can be sorted:

- The SortableColumnAttribute is assigned to the data field's metadata, and its **Sortable** property is `true`.

  [C#]

  ```
  [DES.SortableColumn(true)]
  ```

  [VB]

  ```
  <DES.SortableColumn(True)> _
  ```

- The data field's property type is any of these: integer, floating point, DateTime, TimeSpan, or DateTimeOffset.
- The data field's property type is string and its maximum length has been declared as less than or equal to the constant **PeterBlum.DES.DynamicData.SortableColumnAttribute.SortableMaxLength**, which defaults to 100. The SortableMaxLength can be overridden on the individual DynamicColumnTitle control with its own **SortableMaxLength** property.

If you want to disallow a data field from sorting, apply the SortableColumnAttribute with the Sortable property as false.

[C#]

```
[DES.SortableColumn(false)]
```

[VB]

```
<DES.SortableColumn(False)> _
```

Establishing the DataField name

Assign the column name in the **DataField** property. When used within a Pattern Template, you can leave the **DataField** property unassigned to let the DynamicPattern, DynamicListView, or DynamicFormView control determine its value.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Choosing a source for the column's metadata

There are several sources for text associated with a column. They come from either the DisplayNameAttribute, DescriptionAttribute, or DisplayAttribute. Specify which to use within the **TextSource** and **TextSourceFallback** properties of the DynamicLabel control.

- The **DisplayName** property, defined either by with DisplayNameAttribute or DisplayAttribute.
- The **ShortDisplayName** property, defined either by with DisplayNameAttribute or DisplayAttribute.
- The **Description** property, defined either by with DescriptionAttribute or DisplayAttribute.
- The **Prompt** property, defined either by with DescriptionAttribute or DisplayAttribute.
- The **Other** property of the Label control itself.

Example: Column's description

Note: The tagPrefix "desDD" refers to the PeterBlum.DES.DynamicData assembly and namespace.

```
<desDD:DynamicColumnTitle ID="DynamicColumnTitle2" runat="server"
      TextSource="ShortDisplayName" />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Formatting the text

The **Text** property determines if the column's text appears and if so, where. Whenever it appears, the column's text is determined by the **TextSource** and **TextSourceFallback** properties as described above.

- If the **Text** property is blank, it will be assigned the column's text.
- If the **Text** property is not blank, the "{COLUMN}" token is replaced by the column's text. For example:

```
Text="{COLUMN}:"
```

- If the **Text** property contains "{COLUMN:*column name*}" tokens, they are replaced by the column's text defined in the token. For example:

```
Text="{COLUMN} through {COLUMN:enddate}"
```

- If the TextLookupID property is assigned, it gets the initial value for the Text property from the String Lookup System (see the **General Features Guide**.)

Use the **Case** property to change the case of its characters.

Use the **CssClass** property to specify a style sheet class name.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Sorting Interface Appearance

By default, the interface for a sortable column title looks like this:

Product name

[LinkButton with data field name] [(to switch to descending) or (to switch to ascending)]

Use these properties to customize it:

- **ButtonCssClass** – The style sheet class for the LinkButton.
- **ButtonSeparator** – The HTML between the LinkButton and Image, which defaults to " ".
- **AscendingImageUrl** – A URL to the image used to indicate an ascending sort. It supports mouseover and mousepressed effects by including additional image files with specific names in the same folder.
- **DescendingImageUrl** – A URL to the image used to indicate a descending sort. It supports mouseover and mousepressed effects by including additional image files with specific names in the same folder.
- **ImageAlign** – Adjusts the position of the image.
- **AscendingImageText** – The alternative text and tooltip for the Ascending Image.
- **DescendingImageText** – The alternative text and tooltip for the Descending Image.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties of the DynamicColumnTitle control

The DynamicColumnTitle control subclasses from `PeterBlum.DES.DynamicData.DynamicLabel`, which is an ancestor of the `System.Web.UI.WebControls.Label` control.

Click on any of these topics to jump to them:

- Source of data Properties
- Textual Content Properties
- Additional formatting properties
- Sorting Interface Properties
- Behavior Properties

Sorting Interface Properties

These properties are used when the data field is determined to be sortable. See "Establishing Sorting on a Data Field".

- **ButtonCssClass** (string) – The style sheet class for the LinkButton. (The CssClass property itself is applied to the entire group of LinkButton + ButtonSeparator + Image.)

  You can define mousepressed and mouse over styles by using the same style sheet class name plus the text "Pressed" or "MouseOver". These styles will merge with the class name defined here. So any properties in the pressed and mouseover classes will overrided properties in this, but not the entire style.

  If blank, it is not used.

  To use automatic browser renaming of the class, start with an ! character.

  It defaults to "".

- **ButtonSeparator** (string) – The HTML between the LinkButton and Image. It defaults to "` `".

- **AscendingImageUrl** (string) – A URL to the image used to indicate an ascending sort. Use **AscendingImageText** for the alternative text and tooltip on this image.

  It defaults to `"{APPEARANCE}/Shared/DnArrow1.GIF"`.

  ### Special Symbols for URLs

  The "{APPEARANCE}" token will be replaced by the default path to the **Appearance** folder, which you defined as you set up the web site.

  Supports the use of the tilde (~) as the first character to be replaced by the virtual path to the web application.

  ### Images for Pressed and MouseOver Effects

  You can have images for pressed and mouseover effects as well as the normal image. The names of the image files determine their purpose. Define the name of the normal image. For example, "myimage.gif". Create the pressed version and give it the same name, with "Pressed" added before the extension. For example, "myimage*pressed*.gif". Create the mouseover version and give it the same name, with "MouseOver" added before the extension. For example, myimage*mouseover*.gif.

  The **AscendingImageUrl** 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.*

- **DescendingImageUrl** (string) – A URL to the image used to indicate a descending sort. Use **DescendingImageText** for the alternative text and tooltip on this image.

  It defaults to `"{APPEARANCE}/Shared/DnArrow1.GIF"`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Special Symbols for URLs

The "{APPEARANCE}" token will be replaced by the default path to the **Appearance** folder, which you defined as you set up the web site.

Supports the use of the tilde (~) as the first character to be replaced by the virtual path to the web application.

Images for Pressed and MouseOver Effects

You can have images for pressed and mouseover effects as well as the normal image. The names of the image files determine their purpose. Define the name of the normal image. For example, "myimage.gif". Create the pressed version and give it the same name, with "Pressed" added before the extension. For example, "myimage*pressed*.gif". Create the mouseover version and give it the same name, with "MouseOver" added before the extension. For example, myimage*mouseover*.gif.

The **DescendingImageUrl** 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.*

- **ImageAlign** (enum System.Web.UI.WebControls.ImageAlign) – Adjusts the position of the image. It defaults to `ImageAlign.Top`.
- **AscendingImageText** – The alternative text and tooltip for the Ascending Image. It defaults to "Ascending".
- **AscendingImageTextLookupID** – An alternative to **AscendingImageText**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. Strings must be in the string group Misc.
- **DescendingImageText** – The alternative text and tooltip for the Descending Image. It defaults to "Descending".
- **DescendingImageTextLookupID** – An alternative to **DescendingImageText**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. Strings must be in the string group Misc.

Behavior Properties

- **SortableMaxLength** (int) – If the **DataField** is a string type and is not assigned a SortableColumnAttribute, this is the maximum length of the data field supporting that string that allows sorting.

  Data fields that do not specify their maximum length are ignored by this property and must use the SortableColumnAttribute if you want them to sort.

  When 0, it uses a global value from **SortableColumnAttribute.SortableMaxLength** that is assigned in `Application_Start()` or if unassigned, defaults to 100.

  It defaults to 0, and therefore uses the global default.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

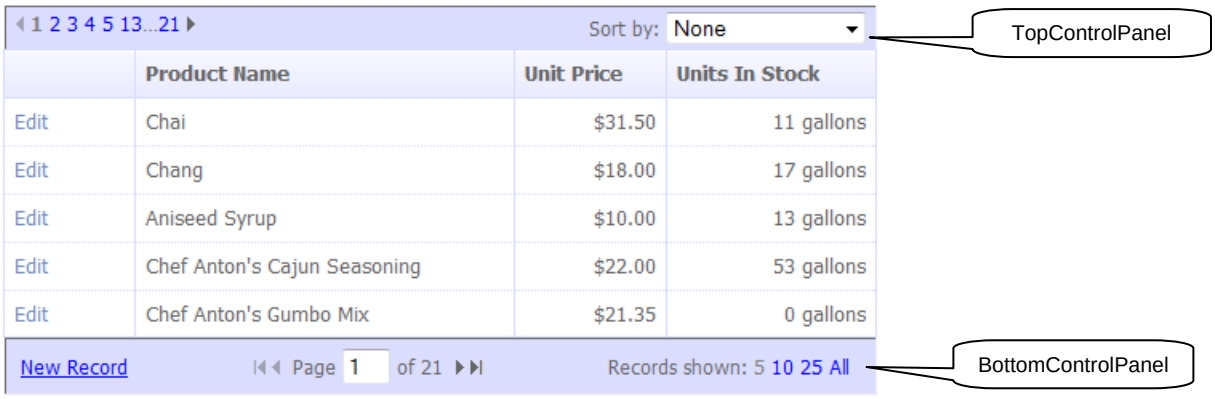
The DynamicControlPanel Control – The Enhanced DataPager

The DynamicControlPanel control subclasses from `System.Web.UI.WebControls.DataPager` to provide a user interface for paging and manipulating the data shown within a Databound Control. The native control lets you define several interfaces for paging in its **Fields** property using these classes: NextPreviousPagerField, NumericPagerField, and TemplatePagerField. It only supports the ListView control.

The DynamicControlPanel expands the available paging user interfaces with many new classes. In addition, it offers data about the paging state, lets the user specify the number of items shown in a list, establish the sort, and add command buttons. It is a "control panel" that manages the Databound Control. The only thing it doesn't handle is filtering; yet you can add that into the container around these tools with its Templating capabilities and the Filtering features of ASP.NET Dynamic Data.

Click on any of these topics to jump to them:

- Features
- Using the DynamicControlPanel
  - Connecting to the DataBound control
  - Fields – Subclasses of BaseDynamicControlPanelField
  - Establishing the appearance for the DynamicControlPanel
- Adding a DynamicControlPanel Control
- Properties of the DynamicControlPanel Control
- Properties on Field Classes
- Constructors of the various Field Classes




```
<desDD:DynamicControlPanel ID="TopControlPanel" runat="server"
  PagedControlID="DynamicListView1" >
  <ContainerTagAttributes Style="height:2em;" />
  <Fields>
    <desDD:DropDownListSorterField>
      <ContainerTagAttributes Style="float:right;" />
    </desDD:DropDownListSorterField>
    <desDD:SpreadExtendedNumericPagerField AlwaysShowFirstLastPageNumbers="True"
      MinimumBetweenPages="5">
    </desDD:SpreadExtendedNumericPagerField>
  </Fields>
</desDD:DynamicControlPanel>
```

BOTTOMCONTROLPANEL EXAMPLE ON THE NEXT PAGE

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

```
<desDD:DynamicControlPanel ID="BottomControlPanel" runat="server"
   PagedControlID="DynamicListView1" PageSize="5" >
   <ContainerTagAttributes Style="height:2em;text-align:center;" />
   <Fields>
      <desDD:NewButtonControlPanelField Text="New Record" >
         <ContainerTagAttributes Style="float:left;" />
      </desDD:NewButtonControlPanelField>
      <desDD:LinksPageSizesField >
         <ContainerTagAttributes Style="float:right;"/>
      </desDD:LinksPageSizesField>
      <desDD:PreviousPagerField />
      <desDD:InfoPagerField />
      <desDD:NextPagerField />
   </Fields>
</desDD:DynamicControlPanel>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Features

It expands upon the native DataPager with these features:

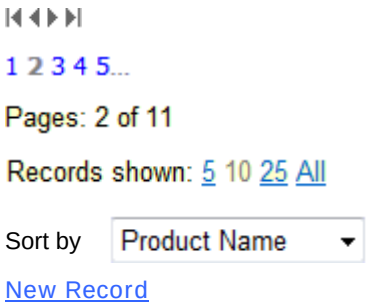
- It introduces more powerful replacements to the existing PagerField classes. It also introduces classes with new features, including sorting, setting the number of record per page, and command buttons. See "Fields – Subclasses of BaseDynamicControlPanelField".
- Support for any DataBound control that has a Databound Control Adapter. That includes GridView, DetailsView, ListView, FormView, DynamicListView, and DynamicFormView.
- It creates an optional HTML control container around the paging and manipulation controls.
- Design mode support for Extended Properties Editor and SmartTags.
- **PagedControlID** can be assigned by control reference in the **PagedControl** property.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the DynamicControlPanel

There are several tasks when using this control:

- Connect the DynamicControlPanel to the DataBound control that it manages. See "Connecting to the DataBound control".
- Selecting which "Fields" to include. A Field is a subclass of `PeterBlum.DES.Dynamic.BaseDynamicControlField` that provides paging, sorting, displaying or other manipulation of the data within the Databound control. See "Fields – Subclasses of BaseDynamicControlPanelField".




```
<desDD:NextPreviousPagerField />
<desDD:NumericPagerField />
<desDD:InfoPagerField />
<desDD:LinksPageSizesField />
<desDD:DropDownSorterField />
<desDD:NewCommandButtonField />
```

- Determining their position. The same Field may appear in several places, such as above and below the DataBound control. Several fields may go side-by-side, such as the InfoPagerField surrounded by PreviousPagerField and NextPagerField, like this:

```
<desDD:PreviousPagerField />
<desDD:InfoPagerField />
<desDD:NextPagerField />
```

- Creating a user interface that contains these fields, such as a `<div>` tag with a style sheet to form a box around a group of fields. The DynamicControlPanel provides this container around multiple Fields. Individual Fields can have their own container, either for its appearance (like a border around it) or positioning (like using `style="float:right;"`).

```
<desDD:DynamicControlPanel ID="BottomControlPanel" runat="server"
   PagedControlID="DynamicListView1" ContainerTag="Div" >
   <ContainerTagAttributes Style="height:2em; text-align:center;" />
   <Fields />
</desDD:DynamicControlPanel>
```

Click on any of these topics to jump to them:

- Connecting to the DataBound control
- Fields – Subclasses of BaseDynamicControlPanelField
- Establishing the appearance for the DynamicControlPanel

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Connecting to the DataBound control

The DynamicControlPanel needs to know which DataBound control it manages. If the control is inside a <PagerTemplate> node of the GridView, DetailsView, ListView, or DynamicListView, it automatically knows the DataBound control.

If the control is outside the DataBound control, set the **ID** of the DataBound control to the **PagedControlID** property.

When working with a list-style DataBound control, set the number of rows shown per page in the **PageSize** property. It defaults to 10.

```
<desDD:DynamicControlPanel ID="BottomControlPanel" runat="server"
   PagedControlID="DynamicListView1" PageSize="5" >
```

Note: If you have multiple DynamicControlPanels on a single DataBound control, set the PageSize on only one of them.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Fields – Subclasses of BaseDynamicControlPanelField

After adding the DynamicControlPanel itself, add one or more objects to its Fields collection. These objects, called Fields, are subclassed from PeterBlum.DES.DynamicData.BaseDynamicControlPanelField. They give various interfaces for paging and manipulating data.

Class	Description
DES.NextPreviousPagerField |◀ ◀ ▶ ▶|	Buttons for moving to next, previous, first, and last "page". You determine the type of button, the layout, and if buttons are omitted.
PreviousPagerField |◀ ◀	A subclass of DES.NextPreviousPagerField that only offers buttons for previous and first "page".
NextPagerField ▶ ▶|	A subclass of DES.NextPreviousPagerField that only offers buttons for next and last "page".
DES.NumericPagerField 1 2 3 4 5... |◀ ◀ 1 2 3 4 5... ▶ ▶| ◀ 1...5 6 7 8 9...52 ▶	A list of page numbers that the user can click to display. It optionally offers the first and last page numbers when the range of page numbers is not near the first and last page. It also offers buttons for moving to next, previous, first, and last "page" as it inherits from DES.NextPreviousPagerField.
IncrementExtendedNumericPagerField ◀ 1 2 3 4 5 15... ▶ ◀ 1 2 3 4 5 15...21 ▶ 1 2 3 4 5 | 10 20...	Based on DES.NumericPagerField, it adds an extended list of page numbers to the start or end of the sequential range. The page numbers have a specific number of pages gap between them. For example, when the gap is 10 and the page numbers shown are 1 2 3 4, the extended numbers are 14 24 34 and 44.
SpreadExtendedNumericPagerField 1 | 5 6 7 8 9 | 31 52 ◀ 1...5 6 7 8 9 | 20 30 41...52 ▶	Based on DES.NumericPagerField, it adds an extended list of page numbers to the start or end of the sequential range. The extended page numbers are spread evenly to fill a gap between the sequential range and the start or end page. For example, if there are 100 pages and the sequential range is showing 1 2 3 4 5, the gap is 95. When you indicate that 5 page numbers will fill the gap, the increment is calculated as 19. The following page numbers are shown: 1 2 3 4 5 24 43 62 81 100
JumpByExtendedNumericPagerField ◀ 1 2 3 4 5 +10... ▶	Based on DES.NumericPagerField, it adds an extended list of page numbers to the start or end of the sequential range. It adds +*n* and –*n*, where *n* is the number of pages to advance forward or back.
InfoPagerField Pages: 2 of 11 Jump to page: 2	Displays information about the current page, including the page index, total pages, total records and much more. It optionally lets the user enter a page number in an IntegerTextBox to jump to that page.
DropDownPageSizesField Records shown: 10 5 10 25 All	A dropdownlist offering a various sizes for the number of items to show per page.
LinksPageSizesField Records shown: 5 10 25 All	A list of hyperlinks offering various sizes for the number of items to show per page.

TABLE CONTINUED ON THE NEXT PAGE

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Class	Description
DropDownSorterField Sort by Product Name	A dropdownlist offering the available sort keys. It is an alternative to providing sorting on column titles.
LinksSorterField Sort by None Product Name Price Units In Stock	A list of hyperlinks offering the available sort keys. It is an alternative to providing sorting on column titles.
CommandButtonField Edit	Provide a button that invokes a command in the Databound control. It can be a Button, Image, or LinkButton.
NewCommandButtonField New Record	Since its so common on a single record view, this is a CommandButtonField setup for the New command (to start a new record). Also consider the DynamicButtonsField, below.
DynamicButtonsField Delete Edit New	Use the DynamicButtons control to provide buttons for these commands: New, Edit, Delete, and Details. Normally DynamicButtons is placed inside the DataBound control, but that limits its placement options and with lists, takes up room on each row. When used with a DynamicListView, set **AllowSelectingRows** to `true` to expose the Edit, Delete, and Details commands when a row is selected.
TemplateControlPanelField	Provides a template to create your own HTML and controls for paging or manipulating the data. It is common to add filter controls or the DynamicButtons control into its Template. *Note: Filters are available from ASP.NET Dynamic Data's toolset.*

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the NextPreviousPagerField

ALERT: This class is not the same as System.Web.UI.WebControls.NextPreviousPagerField. DES's is not subclassed from the native object, and as a result has different capabilities.

The NextPreviousPagerField provides buttons for these paging actions: First page, Previous Page, Next Page, and Last Page. By default, it shows all four buttons using images, in this style: ⏮ ◀ ▶ ⏭

It automatically disables buttons if they are not needed, such as the First button when showing the first page. It also automatically hides itself if there is only one page. (All of these behaviors can be customized.)

To change the appearance of individual buttons, set its **ButtonType** property to Button, Link, or Image. Use the **FirstPageText**, **PreviousPageText**, **NextPageText**, and **LastPageText** properties to adjust their labels. (When using images, these are used for the tooltip and `alt=` attribute.) Use the **FirstPageImageUrl**, **PreviousPageImageUrl**, **NextPageImageUrl**, and **LastPageImageUrl** properties to change the images. If you use **ButtonType**=Link, you can show both text and image based on the LinkButtonLayout property.

The HTML surrounding the buttons and their order are controlled by the string in the Layout property. This string takes the following tokens to represent where buttons go: {FIRST}, {PREV}, {NEXT}, and {LAST}. For example, if you want the first two buttons separated from the last two by a pipe character, use **Layout**=`"{FIRST} {PREV} | {NEXT} {LAST}"`.

These paging commands can avoid posting back to request a page. Instead, they can use a parameter on the URL's querystring, like http://mysite.com?jump=next. To use this technique, set the **QueryStringField** property to the name of the desired querystring parameter. The rest is handled for you. *Alert:The ButtonType property is ignored. Buttons are always output as hyperlinks. If you have setup images, they will be used within the hyperlinks.*

Each Field can have a container that visually encloses it or is used to position it. The container is really an HTML tag, so define the desired HTML tag name in the **ContainerTag** property and any styles and attributes in its **ContainerTagAttributes** property.

You can insert HTML before and after the field by using the **HTMLBefore** and **HTMLAfter** properties. When using a container around the field, this HTML is outside of the container, making it a good separator between fields.

Examples

First Previous Next Last

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:NextPreviousPagerField ButtonType="Button" HTMLAfter="<br />" />
  </Fields>
</desDD:DynamicControlPanel>
```

⏮ ◀ | ▶ ⏭

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:NextPreviousPagerField ButtonType="Link"
      Layout="{FIRST} {PREV} | {NEXT} {LAST}"  />
  </Fields>
</desDD:DynamicControlPanel>
```

⏮ ◀ ▶ ⏭

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:NextPreviousPagerField>
      <ContainerTagAttributes Style="float:right;" />
    </desDD:NextPreviousPagerField>
  </Fields>
</desDD:DynamicControlPanel>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the PreviousPagerField and NextPagerField

These Fields are subclasses of `PeterBlum.DES.DynamicData.NextPreviousPagerField`. They predefine the **Layout** property to show only two of the available four commands. For PreviousPagerField, it shows the First and Previous buttons. Its **Layout** property defaults to "`{FIRST} {PREV}`". For NextPagerField, it shows the Next and Last buttons. Its **Layout** property defaults to "`{NEXT} {LAST}`".

These Fields are often used on opposite sides of other Fields or even against different sides of a container. See the examples.

For other properties, see "Using the NextPreviousPagerField".

Examples

Page 1 of 21

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:PreviousPagerField />
    <desDD:InfoPagerField />
    <desDD:NextPagerField />
  </Fields>
</desDD:DynamicControlPanel>
```

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:PreviousPagerField>
      <ContainerTagAttributes Style="float:left;" />
    </desDD:PreviousPagerField>
    <desDD:NextPagerField>
      <ContainerTagAttributes Style="float:right;" />
    </desDD:NextPagerField>
  </Fields>
</desDD:DynamicControlPanel>
```

If you want to center other fields between these two, add those fields after the list shown and set the container's style="text-align:center;".

Pages: 2 of 11

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1"
  ContainerTag="Div" >
  <ContainerTagAttributes Style="text-align:center;" />
  <Fields>
    <desDD:PreviousPagerField>
      <ContainerTagAttributes Style="float:left;" />
    </desDD:PreviousPagerField>
    <desDD:NextPagerField>
      <ContainerTagAttributes Style="float:right;" />
    </desDD:NextPagerField>
    <desDD:InfoPagerField Layout="Pages: {PAGEINDEX} of {PAGETOTAL}"
      EditablePageIndex="false" />
  </Fields>
</desDD:DynamicControlPanel>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the NumericPagerField

ALERT: This class is not the same as System.Web.UI.WebControls.NumericPagerField. DES's is not subclassed from the native object, and as a result has different capabilities.

This field provides list of page numbers that the user can click to display. 1 2 3 4 5...

The page number buttons are either Hyperlinks or Buttons as determined by the **NumericButtonType** property. The current page is shown a as Label.

The maximum number of buttons (pages) shown is determined by the **MaxPageNumbersShown** property, which defaults to 5. If you want to always show the first and last page numbers, set **AlwaysShowFirstLastPageNumbers** to true. By default, it always shows an ellipsis as an indication of a gap between pages. You can change this indicator or hide it by setting the **GapSeparator** property.

There are several subclasses to NumericPagerField that offer extended numbering schemes. They should be considered instead of this class:

IncrementExtendedNumericPagerField ◀ 1 2 3 4 5 15... ▶ ◀ 1 2 3 4 5 15...21 ▶ 1 2 3 4 5 | 10 20...	Adds an extended list of page numbers to the start or end of the sequential range. The page numbers have a specific number of pages gap between them. For example, when the gap is 10 and the page numbers shown are 1 2 3 4, the extended numbers are 14 24 34 and 44.
SpreadExtendedNumericPagerField 1 | 5 6 7 8 9 | 31 52 ◀ 1...5 6 7 8 9 | 20 30 41...52 ▶	Adds an extended list of page numbers to the start or end of the sequential range. The extended page numbers are spread evenly to fill a gap between the sequential range and the start or end page. For example, if there are 400 pages and the sequential range is showing 1 2 3 4, the gap is 396. When you indicate that 5 page numbers will fill the gap, the increment is calculated as 66. The following 5 page numbers are shown: 70 136 202 268 335.
JumpByExtendedNumericPagerField ◀ 1 2 3 4 5 +10... ▶	Adds an extended list of page numbers to the start or end of the sequential range. It adds +*n* and –*n*, where *n* is the number of pages to advance forward or back.

The NumericPagerField also offers buttons for moving to next, previous, first, and last "page" as it inherits from DES.NextPreviousPagerField. ⏮ ◀ 1 2 3 4 5... ▶ ⏭. It offers the Next and Previous buttons by default.

To add buttons, change the **Layout** property to include the tokens for those buttons: {FIRST}, {PREV}, {NEXT}, and {LAST}. Always include the token {NUMBERS} to represent where the page numbers are shown.

For example: **Layout**="`{FIRST} {PREV} {NUMBERS} {NEXT} {LAST}`".

To remove the buttons, use **Layout**="`{NUMBERS}`".

For other properties that control these additional buttons, see "Using the NextPreviousPagerField".

These paging commands can avoid posting back to request a page. Instead, they can use a parameter on the URL's querystring, like http://mysite.com?jump=11. To use this technique, set the **QueryStringField** property to the name of the desired querystring parameter. The rest is handled for you. *Alert:The ButtonType property is ignored. Buttons are always output as hyperlinks. If you have setup images, they will be used within the hyperlinks.*

Each Field can have a container that visually encloses it or is used to position it. The container is really an HTML tag, so define the desired HTML tag name in the **ContainerTag** property and any styles and attributes in its **ContainerTagAttributes** property.

You can insert HTML before and after the field by using the **HTMLBefore** and **HTMLAfter** properties. When using a container around the field, this HTML is outside of the container, making it a good separator between fields.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Examples

3 4 5 6

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:NumericPagerField MaxPageNumbersShown="4" GapSeparator=""
    Layout="{NUMBERS}" />
  </Fields>
</desDD:DynamicControlPanel>
```

◀1...5 6 7 8 9...52▶

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:NumericPagerField AlwaysShowFirstLastPageNumbers="true" />
  </Fields>
</desDD:DynamicControlPanel>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the IncrementExtendedNumericPagerField

Based on NumericPagerField, it adds an extended list of page numbers to the start or end of the sequential range. The page numbers have a specific number of pages gap between them. ◀ 1 2 3 4 5 15... ▶

For example, when the gap is 10 and the page numbers shown are 1 2 3 4 5, the extended numbers are 15 25 35, etc.

◀ 1 2 3 4 5 15... ▶

Start with the basic NumericPagerField and make sure its setup as needed. See "Using the NumericPagerField".

By default, it will show 5 pages in sequence, with up to 5 extended page numbers separated by 10 pages each. It also shows the Previous and Next button commands.

To change the maximum number of extended page numbers shown, set **ExtMaxPageNumbersShown**. You can specify a minimum number of extended page numbers shown too. Set it in **ExtMinPageNumbersShown**. It defaults to 1.

To change the increment that separates each extended page number, set **Increment**.

If you take no further action, the extended numbers will be added to the last sequential page number shown, so if the increment is 10 and the last sequential page number is 7, you will see 3 4 5 6 7 17 27 37, etc. You may prefer the extended numbers to be rounded to the nearest 10th page, such as 3 4 5 6 7 20 30 50. To do this, set **RoundToNearest** to `true`.

You can show HTML between the sequential and extended ranges to separate them using the **RangesSeparator**. It defaults to " ".

Examples

3 4 5 6 7 17 27 37

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:IncrementExtendedNumericPagerField ExtMaxPageNumbersShown="3"
      Layout="{NUMBERS}" />
  </Fields>
</desDD:DynamicControlPanel>
```

◀ 1 2 3 4 5 15 25...35 ▶

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:IncrementExtendedNumericPagerField
      AlwaysShowFirstLastPageNumbers="true" />
  </Fields>
</desDD:DynamicControlPanel>
```

◀ 1 2 3 4 5 | 10 20 30... ▶

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:IncrementExtendedNumericPagerField RoundToNearest="true"
      ExtMaxPageNumbersShown="3" RangesSeparator=" | " />
  </Fields>
</desDD:DynamicControlPanel>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the SpreadExtendedNumericPagerField

Based on NumericPagerField, it adds an extended list of page numbers to the start or end of the sequential range. The extended page numbers are spread evenly to fill a gap between the sequential range and the start or end page.

◀ 1...5 6 7 8 9 | 20 30 41...52 ▶

For example, if there are 100 pages and the sequential range is showing 1 2 3 4 5, the gap is 95. When you indicate that 5 page numbers will fill the gap, the increment is calculated as 19. The following page numbers are shown:

1 2 3 4 5 24 43 62 81 100

Start with the basic NumericPagerField and make sure its setup as needed. See "Using the NumericPagerField".

By default, it will show 5 pages in sequence, with up to 5 extended page numbers to fill the gaps to the before and after the sequential range. It also shows the Previous and Next button commands.

To change the maximum number of extended page numbers shown, set **ExtMaxPageNumbersShown**. You can specify a minimum number of extended page numbers shown too. Set it in **ExtMinPageNumbersShown**. It defaults to 1.

If the calculated number between pages is less than **MinimumBetweenPages**, fewer page numbers are shown until either they have this distance or there is nothing to shown.

You can show HTML between the sequential and extended ranges to separate them using the **RangesSeparator**. It defaults to " ".

Examples

◀ 1 2 3 4 5 | 21 35 ▶

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:SpreadExtendedNumericPagerField
      ExtMaxPageNumbersShown="2" ExtMinPageNumbersShown="1"
      RangesSeparator=" | " />
  </Fields>
</desDD:DynamicControlPanel>
```

◀ 1 2 3 4 5 | 20...35 ▶

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:IncrementExtendedNumericPagerField RangesSeparator=" | "
      AlwaysShowFirstLastPageNumbers="true" />
  </Fields>
</desDD:DynamicControlPanel>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the JumpByExtendedNumericPagerField

Based on NumericPagerField, it adds an extended list of page numbers to the start or end of the sequential range. It adds +*n* and –*n*, where *n* is the number of pages to advance forward or back. ◀ 1 2 3 4 5 +10... ▶

Start with the basic NumericPagerField and make sure its setup as needed. See "Using the NumericPagerField".

By default, it will show 5 pages in sequence, with -10 before and +10 after the sequential range. It also shows the Previous and Next button commands.

To change the jump, set **JumpBy**. You can specify a second jump with the **SecondJumpBy** property.

The text shown for these jumps is defined in the **JumpBackText** and **JumpForwardText** properties. They can show any label but if you want the values you have in **JumpBy** and **SecondJumpBy**, use the token {JUMPBY} or {SIGNEDJUMPBY}. {JUMPBY} does not show the + or – character.

You can show HTML between the sequential and extended ranges to separate them using the **RangesSeparator**. It defaults to " ".

Examples

◀ 1 2 3 4 5 +10... ▶

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:JumpByExtendedNumericPagerField />
  </Fields>
</desDD:DynamicControlPanel>
```

◀ 1 2 3 4 5 Jump +10 Jump +15... ▶

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:JumpByExtendedNumericPagerField
      JumpForwardText="Jump {SIGNEDJUMPBY}" JumpBackText="Jump {SIGNEDJUMPBY}"
      SecondJumpBy="15" />
  </Fields>
</desDD:DynamicControlPanel>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the InfoPagerField

The InfoPagerField displays information about the current page, including the current page index, total pages, total records, and much more. It provides paging through a textbox where you can enter the desired page number.

By default, it shows the current page, as a textbox, and the total pages. Page 1 of 21

The TextBox is actually an IntegerTextBox with a validator attached. The validator uses the error message from the **RangeErrorMessage** property. It's ErrorFormatter is defined in the **ErrorFormatterSkinID** property. The TextBox can offer an optional hint. Its text is assigned in the **PageIndexHint** property.

If you don't want the current page number to be editable, set **EditablePageIndex** to `false`.

Use the **Layout** property to define the tokens for values you want to see and any HTML in between. Here are the available tokens:

{PAGEINDEX} - The current page index where 1 is the first page. When **EditablePageIndex** is `true`, this is replaced by an IntegerTextBox whose **Columns** property is the number of digits in the total pages to show.

{PAGETOTAL} - The total pages that can be shown.

{PERPAGE} - The number of rows per page allowed. Do not use the left or right bracket characters as literal text.

They are reserved for tokens. Instead, use { and }.

{RECSTART} - The record number of the first record shown on this page.

{RECEND} - The record number of the last record shown on this page.

{RECTOTAL} - Total number of records in the query (across all pages).

{RECSHOWN} - The number of records shown on this page.

{PERCENT} - The percentage of the index / total where Index=1 is 0% and Index=Total = 100%.

Each Field can have a container that visually encloses it or is used to position it. The container is really an HTML tag, so define the desired HTML tag name in the **ContainerTag** property and any styles and attributes in its **ContainerTagAttributes** property.

You can insert HTML before and after the field by using the **HTMLBefore** and **HTMLAfter** properties. When using a container around the field, this HTML is outside of the container, making it a good separator between fields.

Examples

Pages: 1 of 11

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
   <Fields>
      <desDD:InfoPagerField EditablePageIndex="false"
         Layout="Pages: {PAGEINDEX} of {PAGETOTAL}" />
   </Fields>
</desDD:DynamicControlPanel>
```

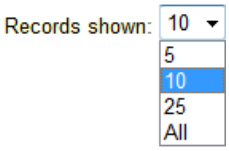
Records: 1 - 10 of total records:104 showing 10 records with limit of 10 per page
0% complete.

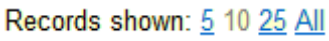
```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
   <Fields>
      <desDD:InfoPagerField Layout="Records: {RECSTART} -
 {RECEND} of total
records:{RECTOTAL} showing {RECSHOWN} records with limit of
{PERPAGE} per page.<br /> {PERCENT}% complete." />
   </Fields>
</desDD:DynamicControlPanel>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the DropDownPageSizesField and LinksPageSizesField

These Fields show a list of sizes for the number of items to show per page. They differ in presentation. DropDownPageSizesField provides a DropDownList, while LinksPageSizesField offers hyperlinks.





By default, they show these values: 5, 10, 25, and "All".

To change the values, define a pipe delimited list in the **Sizes** property. For example, **Sizes**="5|50|100|ALL". Specify the text "ALL" (in uppercase) where the label for "All" will go. If not specified, the "All" option will not be added. The label for the "all" option is assigned to the **AllCommandLabel** property.

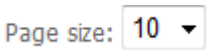
It is not smart to allow the "All" option to include absolutely all records at once, because a typical table has far more records than are appropriate to show. As a result, there is a limit to the size generated by "All". It defaults to 200 and can be changed in the **AllLimit** property.

The HTML around the list of sizes is defined in the **Layout** property. Always include the token {SIZES} to represent the location of the list of sizes. The default text for **Layout** is "Records shown: {SIZES}".

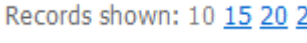
Each Field can have a container that visually encloses it or is used to position it. The container is really an HTML tag, so define the desired HTML tag name in the **ContainerTag** property and any styles and attributes in its **ContainerTagAttributes** property.

You can insert HTML before and after the field by using the **HTMLBefore** and **HTMLAfter** properties. When using a container around the field, this HTML is outside of the container, making it a good separator between fields.

Examples




```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:DropDownPageSizesField Layout="Page size:{SIZES}" AllLimit="100" />
  </Fields>
</desDD:DynamicControlPanel>
```

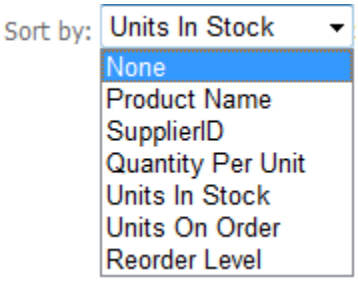



```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:LinksPageSizesField Sizes="10|15|20|25" />
  </Fields>
</desDD:DynamicControlPanel>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the DropDownSorterField and LinksSorterField

These Fields show a list of sort keys. When selected, the page is sorted by the key. They differ in presentation. DropDownSorterField provides a DropDownList, while LinksSorterField offers hyperlinks. When the user selects one, it is shown in ascending order unless you programmatically override it.



Sort by: None Product Name SupplierID Quantity Per Unit Units In Stock Units On Order Reorder Level

By default, they show "None" plus all Sort Keys known by automatic scaffolding to be sortable. That includes any with the SortableColumnAttribute(true) plus these datatypes: integer, floating point, datetime, and strings shorter than a certain limit.

Since scaffolding is part of this process, the list will also omit any field that scaffolding excludes. It will respect the ordering determined by the ScaffoldColumnAttribute's properties.

If you do not use automatic scaffolding, it will still generate in that order unless you override it. To override, set **AutoGenerateSortKeys** to `false` and specify a pipe delimited list of data field names in the **SortKeys** property. The data field names should be their actual names, not the value from the DisplayNameAttribute. The sort keys will be converted to their display name and shown in the order given in the **SortKeys** property. For example, **SortKeys**=`"ProductName|UnitsInStock"`.

The **SortKeys** property is not very flexible. You may prefer to define more complex sort expressions, that involve complex SQL statements and determine the sort order. You can do this programmatically through this method:

[C#]

```
void AddSortKey(string pDisplayName, string pSortExpression,
  PeterBlum.DES.DynamicData.ExpandedSortDirection pSortDirection)
```

[VB]

```
Sub AddSortKey(ByVal pDisplayName As String, ByVal pSortExpression As String,
  ByVal pSortDirection As PeterBlum.DES.DynamicData.ExpandedSortDirection)
```

Parameters

pDisplayName

The text to show the user.

pSortExpression

The sort expression to pass to the DataBound field through its "Sort" command. Often the expression is the data field name (not the displayname) but it may have more complex SQL syntax.

pSortDirection

The Sort Direction. The enumerated type has these values:

- `Ascending` – Always sort in ascending direction.
- `Descending` – Always sort in descending direction.
- `Reverse` – Switch the direction from the current.
- `Current` – Continue with the current direction.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

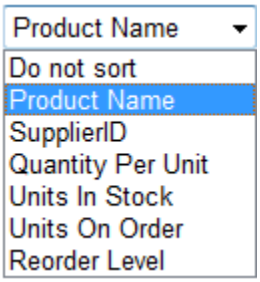
You can offer a "None" option that displays data in unsorted order. Use the **NoSortKeyPosition** property to specify its location – first or last – or if its not shown at all. It shows first by default. The label for the None option comes from the **NoSortKeyLabel** property.

The HTML around the list of sizes is defined in the **Layout** property. Always include the token {SORTKEYS} to represent the location of the list of sort keys. The default text for **Layout** is "`Sort by: {SORTKEYS}`".

Each Field can have a container that visually encloses it or is used to position it. The container is really an HTML tag, so define the desired HTML tag name in the **ContainerTag** property and any styles and attributes in its **ContainerTagAttributes** property.

You can insert HTML before and after the field by using the **HTMLBefore** and **HTMLAfter** properties. When using a container around the field, this HTML is outside of the container, making it a good separator between fields.

Examples




```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:DropDownSorterField Layout="{SORTKEYS}" NoSortKeyLabel="Do not sort" />
  </Fields>
</desDD:DynamicControlPanel>
```

Sort by: Product Name SupplierID Units In Stock

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:LinksSorterField AutoGenerateSortKeys="false"
      SortKeys="ProductName|SupplierID|UnitsInStock" NoSortKeyPosition="None" />
  </Fields>
</desDD:DynamicControlPanel>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the CommandButtonField

The CommandButtonField adds a single button that invokes a command on the DataBound control. The task is trivial when the button is inside of the DataBound control, but the DynamicControlPanel is outside of it. This Field routes the command to the DataBound control. You only need to define the command name in the **CommandName** property and the text to show in the **Text** property.

There are two common cases where other Field classes may be more helpful:

- NewCommandButtonField – Starts a new record.
- DynamicButtonsField – Offers the DynamicButtons control, making it work outside of the DataBound control. It provides these commands: New, Edit, Delete, and Details. It works well with the DynamicListView when you set **AllowSelectingRows** to `true`. (It always works well with single-record DataBound controls.)
- NextPreviousPagerField – Buttons to change the record(s) shown.

The user interface is established with these properties:

- **ButtonType** – Determines if the button is a Button, LinkButton, or ImageButton. Use the values `Button`, `Link` or `Image`. Link supports an image defined in the **ImageUrl** property and/or the **Text**, depending on the LinkButtonLayout property.
- **Text** – The label for the button. For an ImageButton, it is the tooltip and alt= text.
- **ImageUrl** – The URL for the Image.

If your command also takes an argument, set it in the **CommandArgument** property.

Each Field can have a container that visually encloses it or is used to position it. The container is really an HTML tag, so define the desired HTML tag name in the **ContainerTag** property and any styles and attributes in its **ContainerTagAttributes** property.

You can insert HTML before and after the field by using the **HTMLBefore** and **HTMLAfter** properties. When using a container around the field, this HTML is outside of the container, making it a good separator between fields.

Example

New Record

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:NewCommandButtonField Text="New Record" />
  </Fields>
</desDD:DynamicControlPanel>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the NewCommandButtonField

Adds a New command button to the DynamicControlPanel. Another option is to use the DynamicButtonsField, which also offers other command buttons and is smart enough to know when to hide them.

The user interface is established with these properties:

- **ButtonType** – Determines if the button is a Button, LinkButton, or ImageButton. Use the values `Button`, `Link` or `Image`. Link supports an image defined in the **ImageUrl** property and/or the **Text**, depending on the LinkButtonLayout property.
- **Text** – The label for the button. For an ImageButton, it is the tooltip and alt= text.
- **ImageUrl** – The URL for the Image.

Each Field can have a container that visually encloses it or is used to position it. The container is really an HTML tag, so define the desired HTML tag name in the **ContainerTag** property and any styles and attributes in its **ContainerTagAttributes** property.

You can insert HTML before and after the field by using the **HTMLBefore** and **HTMLAfter** properties. When using a container around the field, this HTML is outside of the container, making it a good separator between fields.

Example

New Record

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:NewCommandButtonField Text="New Record" />
  </Fields>
</desDD:DynamicControlPanel>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the DynamicButtonsField

Use the DynamicButtons control to provide buttons for these commands: New, Edit, Delete, and Details. Normally DynamicButtons is placed inside the DataBound control, but that limits its placement options and with lists, takes up room on each row.

This Field exposes the DynamicButtons control as its **DynamicButtons** property. See "Properties for the DynamicButtons Control" for details on its properties.

When working with List style DataBound controls

The New command does not depend on a specific row and therefore is always shown. Edit, Delete, and Details depend on a specific row. As a result, they need to use the ListView's row selection feature (where **SelectedIndex** is a row number).

When used with a DynamicListView, set **AllowSelectingRows** to `true` to expose the Edit, Delete, and Details commands when a row is selected.

Other appearance properties

Each Field can have a container that visually encloses it or is used to position it. The container is really an HTML tag, so define the desired HTML tag name in the **ContainerTag** property and any styles and attributes in its **ContainerTagAttributes** property.

You can insert HTML before and after the field by using the **HTMLBefore** and **HTMLAfter** properties. When using a container around the field, this HTML is outside of the container, making it a good separator between fields.

Example

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:DynamicButtonsField>
      <DynamicButtons ButtonType="Image"
        NewButtonImageUrl="~/Images/NewButton.gif"
        EditButtonImageUrl="~/Images/EditButton.gif"
        DeleteButtonImageUrl="~/Images/DeleteButton.gif" />
    </desDD:DynamicButtonsField>
  </Fields>
</desDD:DynamicControlPanel>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the TemplateControlPanelField

Use the TemplateControlPanel to create custom solutions. It's a great place to add Filter elements, if you want to keep them visually together with other paging and data manipulation elements. If you create a custom paging interface, it can route the paging commands to the DataBound control for you.

Setup your custom solution within the **Template** property. As a template, it allows any HTML and web controls you want. If your Template contains paging controls, set **IncludePagingControls** to `true`.

When your controls invoke commands, like "First" and "Insert", they need to be redirected to the DataBound control for processing. This Field automatically routes these commands: "First", "Prev", "Next", "Last", "PageIndex", and "PageSize". "PageIndex" selects a page by number. It must have the page number (an integer) within the **CommandArgument**. "PageSize" changes the number of rows per page. It must have the number of rows (an integer) within the **CommandArgument**.

```
<desDD:DynamicButtonsField>
<Template>
<des:Button id="PageSize" runat="server" Text="15"
  CommandName="PageSize" CommandArgument="15" />
</Template>
</desDD:DynamicButtonsField>
```

Paging commands are often disabled when they do not apply. Let this Field disable your commands by setting **AutoDisable** to `true`.

If you need to handle other commands or want to change how these are handled, use the PagerCommand event handler. Its Argument parameter contains the existing paging state values in its PagingState property. If your PagerCommand handles the command itself, return Argument.Handled = true.

Here is an example. It assumes there is a textbox called PageNumberTextBox in the Template.

[C#]

```
protected void TemplatePagerField_OnPagerCommand(object sender,
      PeterBlum.DES.DynamicData.DataPagerCommandEventArgs pArgs)
{ // Get the new page number
   System.Web.UI.WebControls.TextBox PageNumberTextBox =
(System.Web.UI.WebControls.TextBox)pArgs.Item.FindControl("PageNumberTextBox");
   int newPageNumber = -1;
   try
   {
      newPageNumber = Convert.ToInt32(PageNumberTextBox.Text.Trim());
   }
   catch (FormatException)
   {
      pArgs.Handled = true;   // so no further action is taken
      return;
   }
   catch (OverflowException)
   {
      pArgs.Handled = true;   // so no further action is taken
      return;
   }

   int vNewIndex = (newPageNumber - 1) * pArgs.PagingState.MaximumRows;
   //Verify if the new index is valid
   if ((vNewIndex >= 0) && (vNewIndex <= pArgs.PagingState.TotalRowCount))
   {
      pArgs.NewStartRowIndex = vNewIndex;
   }
   else
      pArgs.Handled = true;   // so no further action is taken
}
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

[VB]

```
Protected Sub TemplatePagerField_OnPagerCommand(ByVal sender As Object, _
  ByVal pArgs As PeterBlum.DES.DynamicData.DataPagerCommandEventArgs)
    ' Get the new page number
    Dim PageNumberTextBox As System.Web.UI.WebControls.TextBox = _
     DirectCast(pArgs.Item.FindControl("PageNumberTextBox"), _
        System.Web.UI.WebControls.TextBox)
    Dim newPageNumber As Integer = -1
    Try
        newPageNumber = Convert.ToInt32(PageNumberTextBox.Text.Trim())
    Catch generatedExceptionName As FormatException
        pArgs.Handled = True ' so no further action is taken
        Exit Sub
    Catch generatedExceptionName As OverflowException
        pArgs.Handled = True ' so no further action is taken
        Exit Sub
    End Try

    Dim vNewIndex As Integer = (newPageNumber - 1) * pArgs.PagingState.MaximumRows
    'Verify if the new index is valid
    If (vNewIndex >= 0) AndAlso (vNewIndex <= pArgs.PagingState.TotalRowCount) Then
        pArgs.NewStartRowIndex = vNewIndex
    Else
        pArgs.Handled = True ' so no further action is taken
    End If
End Sub
```

If the contents of your template must be changed based on the paging state (like disabling the First command when on the first page), use the **ItemCreated** and **PagingStateChanged** event handlers. **ItemCreated** is called after the Template has been converted into controls. **PagingStateChanged** is fired if the DataBound control had its paging state changed, such as a different page is viewed or a different page size.

Here is an example of both. It assumes there are 3 page size buttons, PageSize5, PageSize10, and PageSizeAll. One should be disabled based on the current page size.

[C#]

```
protected void TemplatePagerField_ItemCreated(object pSender,
      PeterBlum.DES.DynamicData.PagerFieldEventArgs pArgs)
{
   TemplatePagerField_ChangeAppearance(pSender, pArgs);
}
protected void TemplatePagerField_PagingStateChanged(object pSender,
      PeterBlum.DES.DynamicData.PagerFieldEventArgs pArgs)
{
   TemplatePagerField_ChangeAppearance(pSender, pArgs);
}
protected void TemplatePagerField_ChangeAppearance(object pSender,
      PeterBlum.DES.DynamicData.PagerFieldEventArgs pArgs)
{
   WebControl vControlToModify;
   switch (pArgs.PagingState.MaximumRows)
   {
      case 5:
         vControlToModify = (WebControl) pArgs.FindControl("PageSize5");
         vControlToModify.Enabled = false;
         break;
      case 10:
         vControlToModify = (WebControl) pArgs.FindControl("PageSize10");
         vControlToModify.Enabled = false;
         break;
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

```
      case 200:
         vControlToModify = (WebControl) pArgs.FindControl("PageSizeAll");
         vControlToModify.Enabled = false;
         break;
   }
}
```

[VB]

```
Protected Sub TemplatePagerField_ItemCreated(ByVal pSender As Object, _
  ByVal pArgs As PeterBlum.DES.DynamicData.PagerFieldEventArgs)
    TemplatePagerField_ChangeAppearance(pSender, pArgs)
End Sub
Protected Sub TemplatePagerField_PagingStateChanged(ByVal pSender As Object, _
  ByVal pArgs As PeterBlum.DES.DynamicData.PagerFieldEventArgs)
    TemplatePagerField_ChangeAppearance(pSender, pArgs)
End Sub
Protected Sub TemplatePagerField_ChangeAppearance(ByVal pSender As Object, _
  ByVal pArgs As PeterBlum.DES.DynamicData.PagerFieldEventArgs)
  Dim vControlToModify As WebControl
  Select Case pArgs.PagingState.MaximumRows
     Case 5
     vControlToModify = DirectCast(pArgs.FindControl("PageSize5"), WebControl)
     vControlToModify.Enabled = False
     Exit Select
     Case 10
     vControlToModify = DirectCast(pArgs.FindControl("PageSize10"), WebControl)
     vControlToModify.Enabled = False
     Exit Select
     Case 200
     vControlToModify = DirectCast(pArgs.FindControl("PageSizeAll"), WebControl)
     vControlToModify.Enabled = False
     Exit Select
  End Select
End Sub
```

Each Field can have a container that visually encloses it or is used to position it. The container is really an HTML tag, so define the desired HTML tag name in the **ContainerTag** property and any styles and attributes in its **ContainerTagAttributes** property.

You can insert HTML before and after the field by using the **HTMLBefore** and **HTMLAfter** properties. When using a container around the field, this HTML is outside of the container, making it a good separator between fields.

EXAMPLE ON THE NEXT PAGE

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Example

This example uses the event handlers shown on the previous pages.

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
  <desDD:TemplateControlPanelField
    OnPagerCommand="TemplatePagerField_OnPagerCommand"
    OnAfterCreatingControls="TemplatePagerField_ItemCreated"
    OnPagingStateChanged="TemplatePagerField_PagingStateChanged"
    IncludesPagingControls="true">
    <Template>
      Page
      <asp:Label runat="server" ID="CurrentPageLabel"
        Text="<%# Container.TotalRowCount > 0 ? (Container.StartRowIndex /
               Container.PageSize) + 1 : 0 %>" />
      of
      <asp:Label runat="server" ID="TotalPagesLabel"
        Text="<%# Math.Ceiling ((double)Container.TotalRowCount  /
             Container.PageSize) %>" />
      Jump to page:
      <asp:TextBox ID="PageNumberTextBox" runat="server"  AutoPostBack="true"
        Width="30px"
        Text="<%# Container.TotalRowCount > 0 ? (Container.StartRowIndex /
             Container.PageSize) + 1 : 0 %>" />
      <br /><br />
      <asp:Button ID="PrevButton" runat="server" Text="Previous"
    CommandName="Prev" />
      |
      <asp:Button ID="NextButton" runat="server" Text="Next" CommandName="Next" />
      Page sizes:
      <asp:LinkButton ID="PageSize5" runat="server" Text="5"
    CommandName="PageSize" CommandArgument="5" EnableViewState=false />
      |
      <asp:LinkButton ID="PageSize10" runat="server" Text="10"
    CommandName="PageSize" CommandArgument="10" EnableViewState=false />
      |
      <asp:LinkButton ID="PageSizeAll" runat="server" Text="All"
    CommandName="PageSize" CommandArgument="200" EnableViewState=false />
    </Template>
  </desDD:TemplateControlPanelField>
</Fields>
</desDD:DynamicControlPanel>
```

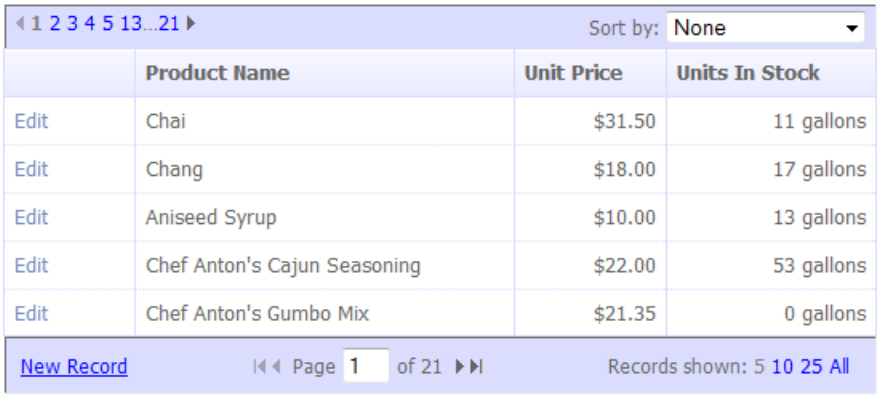
Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Establishing the appearance for the DynamicControlPanel

Initially the DynamicControlPanel control appears to be a way to show paging and other data manipulation features. Yet those features – the Fields – need to be displayed amongst other controls and HTML on the page. Consider these issues:

- There is a container for Fields. It is positioned on a specific side of the DataBound control.
- There may be multiple containers on different sides with different lists of Fields. For example, on the top you have a NextPreviousPagerField and LinksSorterField. On the bottom you have the NewCommandButtonField, NextPreviousPagerField, and LinksPageSizesField.
- Several Fields appear side-by-side while others need to be in different sides of a container. For example, the NextPreviousPagerField appears on the left side and the DropDownPageSizesField appears on the right side.
- The container of the Fields needs its own style sheets to establish border, fonts, and background.
- Individual Fields need styles to assist in their positioning. Common styles are float, width, white-space:nowrap; margin, and padding.

This image has two DynamicControlPanels, on top and bottom. Each has a different list of Fields. Markup follows.




```
<desDD:DynamicControlPanel ID="TopControlPanel" runat="server"
   PagedControlID="DynamicListView1" >
   <ContainerTagAttributes CssClass="ControlPanel" />
   <Fields>
      <desDD:DropDownListSorterField>
         <ContainerTagAttributes Style="float:right;" />
      </desDD:DropDownListSorterField>
      <desDD:SpreadExtendedNumericPagerField AlwaysShowFirstLastPageNumbers="True"
         MinimumBetweenPages="5">
      </desDD:SpreadExtendedNumericPagerField>
   </Fields>
</desDD:DynamicControlPanel>
```

```
<desDD:DynamicControlPanel ID="BottomControlPanel" runat="server"
   PagedControlID="DynamicListView1" PageSize="5" >
   <ContainerTagAttributes CssClass="ControlPanel" />
   <Fields>
      <desDD:NewButtonControlPanelField Text="New Record" >
         <ContainerTagAttributes Style="float:left;" />
      </desDD:NewButtonControlPanelField>
      <desDD:LinksPageSizesField >
         <ContainerTagAttributes Style="float:right;"/>
      </desDD:LinksPageSizesField>
      <desDD:PreviousPagerField />
      <desDD:InfoPagerField />
      <desDD:NextPagerField />
   </Fields>
</desDD:DynamicControlPanel>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Looking through the ASP.NET markup of the previous example, notice the `<ContainerTagAttributes>` nodes. The **ContainerTagAttributes** property holds style sheet class and element-specific styles for the entire DynamicControlPanel and individual Fields.

```
<desDD:DynamicControlPanel ID="BottomControlPanel" runat="server"
  PagedControlID="DynamicListView1" PageSize="5" >
  <ContainerTagAttributes CssClass="ControlPanel" />
  <Fields>
    <desDD:NewButtonControlPanelField Text="New Record" >
      <ContainerTagAttributes Style="float:left;" />
    </desDD:NewButtonControlPanelField>
```

Container Tag and styling on DynamicControlPanel

Use the **ContainerTagAttributes** on DynamicControlPanel to establish the style of the overall control. The DynamicControlPanel outputs an HTML tag, defined in its **ContainerTag** property, and the style you supplied in **ContainerTagAttributes**.

The default of **ContainerTag** is `Auto`. That means it determines its tag based on its location. If it's inside the PagingTemplate of a GridView or DetailsView, it will use a `<tr>` tag and the Fields will use the `<td>` tag because GridView and DetailsView are `<table>` oriented controls. If its inside a DynamicListView's PagingTemplate and DynamicListView.ContainerTag is Table, DynamicControlPanel uses `<tr>` too. When not inside a PagingTemplate, it always uses a `<div>` tag.

Change the **ContainerTag** if the HTML tag output is incorrect.

The **ContainerTagAttributes** can handle three values. Use its **CssClass** property to establish a style sheet class. Use its **Style** property to establish CSS style properties. Usually the **Style** property contains formatting properties that customize the style sheet class further, such as shown here.

```
<desDD:DynamicControlPanel ID="BottomControlPanel" runat="server"
  PagedControlID="DynamicListView1" ContainerTag="Div" >
  <ContainerTagAttributes CssClass="ControlPanel" Style="width:500px;" />
```

If that HTML tag also needs other attributes, like `<table cellspacing="1">`, add the attributes as properties like this:

```
<desDD:DynamicControlPanel ID="BottomControlPanel" runat="server"
  PagedControlID="DynamicListView1" ContainerTag="Table" >
  <ContainerTagAttributes CssClass="ControlPanel" cellspacing="1" />
```

Container Tag and styling on Fields

The same two properties, **ContainerTag** and **ContainerTagAttributes**, are on each Field. While you can use the attributes to provide a different border, font, or background, they are mostly used for formatting.

The default of **ContainerTag** is `Auto`. It determines its value based on the **DynamicControlPanel.ContainerTag**.

DynamicControlPanel's ContainerTag	Field uses
`<tr>`	`<td>`
`<ul>` or `<ol>`	`<li>`
All others	`<div>`

Change the **ContainerTag** if the HTML tag output is incorrect.

You may want several Fields to be enclosed by a common container tag. If so, set the same value for **ContainerTag** on the first and last Field of the group. Set **ContainerTag** to `None` for any inbetween. On the first Field, set **ContainerTagUsage**=`OpeningTag`. On the last Field, set **ContainerTagUsage**=`ClosingTag`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

The **ContainerTagAttributes** can handle three values. Use its **CssClass** property to establish a style sheet class. Use its **Style** property to establish CSS style properties. Usually the **Style** property contains formatting properties that customize the style sheet class further, such as shown here.

```
<desDD:NextPreviousPagerField >
  <ContainerTagAttributes CssClass="NextPrevPager" Style="float:right;" />
```

See also “Style Suggestions”.

If that HTML tag also needs other attributes, like `<td colspan="3">`, add the attributes as properties like this:

```
<desDD:NextPreviousPagerField ContainerTag="TD" >
  <ContainerTagAttributes CssClass="NextPrevPager" colspan="3" />
```

You can place HTML before and after the container tag using the **HTMLBefore** and **HTMLAfter** properties. Use these for HTML that separates the Fields from each other and from the DynamicControlPanel’s container.

```
<desDD:NextPreviousPagerField HTMLBefore=" " HTMLAfter="<br />" />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Additional Behavior Properties of the DynamicControlPanel

- **QueryStringField** – Determines the name of the querystring parameter on the URL that invokes paging commands. When assigned, paging commands are setup using hyperlinks instead of buttons that post back.
- **CausesValidation** – By default, the Fields do not invoke validation on a row being added or edited in the DataBound control. If you prefer to validate, set this to true. The ValidationGroup is automatically setup for you.
- **EditModeBehavior** – When the DataBound control is in Insert or Edit mode, should certain Fields still be shown? For example, should paging controls be hidden? Use this property to hide some or all Fields.
- **OnePageBehavior** – When there is only one row available, should certain Fields still be shown? Use this property to hide some or all Fields.
- **DisableButtonsOnSubmit** – When true, buttons within the DynamicControlPanel are disabled when the page is submitted.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Adding a DynamicControlPanel Control

These steps ask you to jump around the document using clicks on links. Adobe Reader offers a **Previous View** command to return to the link. Look for this in the Adobe Reader (shown v6.0)

1. Prepare the page for DES controls. See "Preparing a page for DES controls" in the **General Features Guide**. It covers issues like style sheets, AJAX, and localization.
2. Add an DynamicControlPanel control to the page. It can go in the <PageTemplate> node of a GridView, DetailsView, ListView or DynamicListView. It also can go outside the DataBound control wherever you prefer.

   Visual Studio and Visual Web Developer Design Mode Users

   Drag the DynamicControlPanel control from the Toolbox onto your web form.

   Text Entry Users

   Add the control:

```
<desDD:DynamicControlPanel id="[YourControlID]" runat="server" />
```

   Programmatically creating the DynamicControlPanel control

   - Identify the control which you will add the DynamicControlPanel control to its **Controls** collection. Like all ASP.NET controls, the DynamicControlPanel can be added to any control that supports child controls, including Templates.
   - Create an instance of the DynamicControlPanel control class. The constructor takes no parameters.
   - Assign the **ID** property.
   - Add the DynamicControlPanel control to the **Controls** collection.

   In this example, the DynamicControlPanel is created with an **ID** of "DynamicControlPanel1". It is added to PlaceHolder1.

   [C#]

```
PeterBlum.DES.DynamicData.DynamicControlPanel vDynamicControlPanel =
   new PeterBlum.DES.DynamicData.DynamicControlPanel();
vDynamicControlPanel.ID = "DynamicControlPanel1";
PlaceHolder1.Controls.Add(vDynamicControlPanel);
```

   *Note: The namespace for these controls is PeterBlum.DES. If you prefer, add a **using** clause to that namespace on your form.*

   [VB]

```
Dim vDynamicControlPanel As PeterBlum.DES.DynamicData.DynamicControlPanel = _
   New PeterBlum.DES.DynamicData.DynamicControlPanel()
vDynamicControlPanel.ID = "DynamicControlPanel1"
PlaceHolder1.Controls.Add(vDynamicControlPanel)
```

   *Note: The namespace for these controls is PeterBlum.DES.DynamicData. If you prefer, add an **Imports** clause to that namespace on your form.*

CONTINUED ON THE NEXT PAGE

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

3. When outside of the DataBound control, set the **PagingControlID** property to the ID of the DataBound control.

> **Guidelines for setting properties**
>
> - Design mode users can use the Properties Editor or the Expanded Properties Editor. (See "Expanded Properties Editor" in the **General Features Guide**.) The SmartTag also offers some of the most important properties.
> - Text entry users should add the properties into the `<desDD:ControlClass>` tag in this format: `propertyname="value"`
> - When setting a property programmatically, have a reference to the control's object and set the property according to your language's rules.

4. When using a list style DataBound control, set the default number of rows shown in the **PageSizes** property. (It defaults to 10.) *Only do this to one DynamicControlPanel per DataBound control.*

5. Add Field objects to the **Fields** property. See "Fields – Subclasses of BaseDynamicControlPanelField".

Visual Studio and Visual Web Developer Design Mode Users

In the SmartTag, select the **Edit Fields** command.

In the Properties Editor, click on the button to the right of the **Fields** property.

The Editor dialog shows a list of Fields. You can add a field with the **Add** button at the bottom. It pops up a list of available Field classes. Click on an existing element in the list to show it in the PropertyGrid for editing.

Click **OK** to save.

Text Entry Users

The ASP.NET markup syntax looks like this:

```
<desDD:DynamicControlPanel ID="DynamicControlPanel1" runat="server"
  PagedControlID="DynamicListView1" >
  <Fields>
    <desDD:FieldClass />
    <desDD:FieldClass Properties >
      <ContainerTagAttributes Properties />
    </desDD:FieldClass >
  </Fields>
</desDD:DynamicControlPanel>
```

Programmatically creating the DynamicControlPanel control

Each Field class has several constructors to make their creation easier. See "Constructors of the various Field Classes".

Instantiate the object and add it to the **Fields** property like this:

[C#]

```
PeterBlum.DES.DynamicData.NextPreviousPagerField vField =
   new PeterBlum.DES.DynamicData.NextPreviousPagerField(ButtonType.Link);
DynamicControlPanel1.Fields.Add(vField);
```

[VB]

```
Dim vField As PeterBlum.DES.DynamicData.NextPreviousPagerField = _
   New PeterBlum.DES.DynamicData.NextPreviousPagerField(ButtonType.Link)
DynamicControlPanel1.Fields.Add(vField)
```

6. Determine the appearance of the DynamicControlPanel's containing HTML tag by setting the **ContainerTag** and **ContainerTagAttributes** properties. See "Container Tag and styling on DynamicControlPanel".

7. If any Field needs adjustments to its positioning or formatting, change the style sheet and style attributes on its own **ContainerTagAttributes** property. See "Container Tag and styling on Fields".

CONTINUED ON THE NEXT PAGE

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

8. Consider other behavior properties on the DynamicControlPanel. See "Additional Behavior Properties of the DynamicControlPanel".

9. Here are some other considerations:

   - If you are using an AJAX system to update this control, set the **InAJAXUpdate** property to `true`. Also make sure the PageManager control or AJAXManager object has been setup for AJAX. See "Using these Controls With AJAX" in the **General Features Guide**. *Failure to follow these directions can result in incorrect behavior and javascript errors.*

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties of the DynamicControlPanel Control

The `PeterBlum.DES.DynamicData.DynamicControlPanel` control is subclassed from `System.Web.UI.WebControls.DataPager`.

Click on any of these topics to jump to them:

- Primary Properties
- Container Appearance Properties
- Behavior Properties
- Properties on Field Classes

Primary Properties

- **PagingControlID** (string) – The ID to the pageable DataBound control. It is optional. When not assigned, the control should be in a PagingTemplate of the DataBound control. This control searches its containing controls to find the DataBound control.

  This is ID property, not the ClientID or UniqueID. It will be translated into the ClientID at runtime. If the control cannot be found in the current or any parent NamingContainer, an exception is thrown at runtime. When assigning programmatically, set it during the `Page_Init()` phase (OnInit method).

- **PagingControl** (System.Web.UI.Control) – An alternative to the **PagingControlID**. A reference to an actual control object. It must be a DataBound control. As a reference to the control, it can refer to any control on the page, whereas the **PagedControlID** property must refer to a control within the same or parent Naming Container as this DynamicControlPanel.

  When assigning programmatically, set it during the `Page_Init()` phase (OnInit method).

- **PageSize** (integer) – Determines the number of records are shown per page. Set this on a list style interface. It defaults to 10.

  Note: If your DataBound control has multiple DynamicControlPanels, only set this on one of them.

- **Fields** (System.Web.UI.WebControls.DataPagerFieldCollection) – A list of Fields to show within this DynamicControlPanel. While it supports any native DataPagerField class, consider using DES's BaseControlPanelField subclasses.

  See "Fields – Subclasses of BaseDynamicControlPanelField".

  ### Visual Studio and Visual Web Developer Design Mode Users

  In the SmartTag, select the **Edit Fields** command.

  In the Properties Editor, click on the button to the right of the **Fields** property.

  The Editor dialog shows a list of Fields. You can add a field with the **Add** button at the bottom. It pops up a list of available Field classes. Click on an existing element in the list to show it in the PropertyGrid for editing.

  Click **OK** to save.

  ### Text Entry Users

  The ASP.NET markup syntax looks like this:

```
<desDD:DynamicControlPanel ID="DynamicControlPanel1" runat="server"
  PagedControlID="DynamicListView1" >
  <Fields>
    <desDD:FieldClass />
    <desDD:FieldClass Properties >
      <ContainerTagAttributes Properties />
    </desDD:FieldClass >
  </Fields>
</desDD:DynamicControlPanel>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Programmatically creating the DynamicControlPanel control

Each Field class has several constructors to make their creation easier. See "Constructors of the various Field Classes".

Instantiate the object and add it to the **Fields** property like this:

[C#]

```
PeterBlum.DES.DynamicData.NextPreviousPagerField vField =
   new PeterBlum.DES.DynamicData.NextPreviousPagerField(ButtonType.Link);
DynamicControlPanel1.Fields.Add(vField);
```

[VB]

```
Dim vField As PeterBlum.DES.DynamicData.NextPreviousPagerField = _
   New PeterBlum.DES.DynamicData.NextPreviousPagerField(ButtonType.Link)
DynamicControlPanel1.Fields.Add(vField)
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Container Appearance Properties

These properties are shown in the "Appearance" category in the Properties window.

- **ContainerTag** (enum PeterBlum.DES.DynamicData.DynamicControlPanelContainerTag) – This control is optionally enclosed in an HTML tag. This property determines the tag type. When used, the **ContainerTagAttributes** property provides style and other attributes to the tag.

  The enumerated type DynamicControlPanelContainerTag has these values:

  - `None` – Do not enclose the Fields in an HTML container.
  - `Auto` – Attempt to sense the correct container from the IPagedItemContainer control, so long as the DynamicControlPanel is inside the paging control, such as in the PagingTemplate of GridView.

    It will use `TableRowWithOneCell` if the container is GridView, DetailsView, DynamicListView with **ContainerTag** defining a `<table>` tag, or DynamicFormView. If DynamicListView.ContainerTag defines a `<ul>` or `<ol>` tag, it will use the `<li>` tag.

    For all others, it uses `<div>`.
  - `Span` – The `<span>` tag.
  - `Div` – The `<div>` tag.
  - `TableRow` – The `<tr>` tag. The Fields must supply the `<td>` tags in their own **ContainerTag** properties.
  - `TableRowWithOneCell` – A `<tr>` tag and one `<td>` cell with the attribute `colspan="9999"`.
  - `TableCell` – The `<td>` tag.
  - `TableHeaderCell` – The `<th>` tag.
  - `Fieldset` – The `<fieldset>` tag.
  - `UL` – The `<ul>` tag
  - `OL` – The `<ol>` tag
  - `LI` – The `<li>` tag

  It defaults to `DynamicControlPanelContainerTag.Auto`.

- **ContainerTagAttributes** (PeterBlum.DES.DynamicData.HTMLTagAttributes) – The `style=`, `class=`, and other attributes applied to the containing HTML tag that is specified in the **ContainerTag** property. It is not used when **ContainerTag** = `None`.

  When **ContainerTag** = `TableRowWithOneCell`, the style is applied to the `<tr>` tag.

  It sets **CssClass** to "`ControlPanel_ContainerTag`" by default, which is found in **~/DynamicData/Content/DESDynamicDataControls.css**.

  This property is defined in the ASP.NET markup like this:

```
<desDD:DynamicControlPanel ID="DynamicControlPanel1" runat="server"
  PagedControlID="DynamicListView1" ContainerTag="Div" >

  <ContainerTagAttributes CssClass="classname" Style="css properties"
    AttributeName="AttributeValue" />

  <Fields> various fields </Fields>

</desDD:DynamicControlPanel>
```

Setting the Style Sheet Class

Define a style sheet class to describe the borders, font, and background of the container around the Fields. Assign its name to the **CssClass** property on **ContainerTagAttributes**.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Setting Style Properties

Use the **Style** property on the **ContainerTagAttributes** property to declare a list of cascading style sheet styles in the CSS syntax. They are assigned to the HTML tag within the `style=` attribute.

The CSS syntax is this pattern:

```
propertyname:value; propertyname2:value2;
```

For example:

```
vertical-align:bottom; white-space:nowrap
```

The CSS property name and value must be identical to what the browser expects or they will not be used.

You can locate the CSS properties here: https://developer.mozilla.org/en/CSS_Reference

When entering the **Style** property in ASP.NET markup, Intellisense is available to assist you in picking style names and defining their values.

When working programmatically, the **Style** property can be assigned in two ways:

To add a single string containing the CSS syntax that replaces the current value of Style, do this:

```
DynamicControlPanel1.ContainerTagAttributes.Style.Value =
"propertyname:value; propertyname2:value2;"
```

To add a single property, do this:

```
DynamicControlPanel1.ContainerTagAttributes.Style.Add("propertyname", "value")
```

Setting other HTML tag attributes

When the ContainerTag's HTML tag needs other attributes than style= and class=, enter them as follows.

- Using ASP.NET Markup, enter them as properties in the `<ContainerTagAttributes>` node like this:

```
<desDD:DynamicControlPanel ID="BottomControlPanel" runat="server"
       PagedControlID="DynamicListView1" ContainerTag="Table" >
    <ContainerTagAttributes CssClass="ControlPanel" cellspacing="1" />
    <Fields> various fields </Fields>
</desDD:DynamicControlPanel>
```

  The property name should be exactly as you want to output into the HTML tag. If you are using XHTML syntax, remember to keep the property name lowercase.

- When programmatically adding attributes, call the SetAttribute() method on the ContainerTagAttributes property like this:

```
DynamicControlPanel1.ContainerTagAttributes.SetAttribute(
    "property name", "value")
```

  For example:

```
DynamicControlPanel1.ContainerTagAttributes.SetAttribute(
    "cellspacing", "1")
```

- **ContainerTagTitle** (string) – Provides a tooltip or `<legend>` tag (when using `<fieldset>`) for the **ContainerTag**.

  For all **ContainerTags**, it supplies the tooltip (`title=` attribute). For fieldset, it also supplies the legend. It is not used when **ContainerTag** = `None`.

  It defaults to "".

- **ContainerTagTitleLookupID** (string) – Gets the value for **ContainerTagTitle** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **ContainerTagTitle** will be used.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Behavior Properties

These properties are shown in the "Behavior" category in the Properties window.

- **InAJAXUpdate** (Boolean) – When using AJAX on this page, set this to `true` if the control is involved in an AJAX update. See "Using These Controls with AJAX" in the **General Features Guide**. It defaults to `false`.
- **Visible** (Boolean) – When `false`, no HTML is output. This control is entirely unused. It defaults to `true`.
- **QueryStringField** (string) – Determines the name of the querystring parameter on the URL that invokes paging commands. This property is used by any Field class that offers commands to change the current page. Other Fields will use a post back to process their requests.

  Using this option overrides AJAX as it requests a page through a URL instead of AJAX.

  When setup, the NextPreviousPagerField and its subclasses only use a hyperlink for the First, Previous, Next and Last button commands. The **ButtonType** property is ignored. Images are still supported if setup as hyperlinks can embed images.

  It defaults to "".
- **CausesValidation** (boolean) – Determines if buttons within the Fields will invoke validation of data within the DataBound control prior to submitting the page. It uses the Validation Group in the **ValidationGroup** property.

  It defaults to `false`.
- **ValidationGroup** (string) – When **CausesValidation** is `true`, validation is invoked against this validation group when the page is submitted.

  Those that match the value in this will be run. When set to "{DEFAULT}", it uses the same validation group as the DataBound control is using in edit or insert mode, so it only interacts with controls inside the DataBound control.

  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 list style DataBound control, 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 "{DEFAULT}".
- **EditModeBehavior** (enum PeterBlum.DES.DynamicData.ModeBehavior) – Determines the appearance of this control or specific ControlPanelFields when the DataBound control is in edit or insert mode.

  GridView will always cause paging controls to disable, regardless of this setting.

  The enumerated type ModeBehavior has these values:

  - `None` – No changes to the DynamicControlPanel. This is the default.
  - `DisablePagingFields` – Disable paging elements in the Fields.
  - `DisableAll` – Disable all controls in the Fields.
  - `HidePagingFields` – Hide paging elements in the Fields.
  - `HideAllFields` – Hide all Fields
  - `HideControlPanel` – Hide the DynamicControlPanel itself.
  - `HTMLWhenFieldNotShownPaging` – Switch to the Field.HTMLWhenFieldNotShown property on paging Fields
  - `HTMLWhenFieldNotShownAllFields` – Switch to the Field.HTMLWhenFieldNotShown property on all Fields.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **OnePageBehavior** (enum PeterBlum.DES.DynamicData.ModeBehavior) – Determines the appearance of this control or specific Fields when the DataBound control can only show a single page.

  GridView will always cause paging controls to disable, regardless of this setting.

  The enumerated type ModeBehavior has these values:

  - `None` – No changes to the DynamicControlPanel. This is the default.
  - `DisablePagingFields` – Disable paging elements in the Fields.
  - `DisableAll` – Disable all controls in the Fields.
  - `HidePagingFields` – Hide paging elements in the Fields.
  - `HideAllFields` – Hide all Fields
  - `HideControlPanel` – Hide the DynamicControlPanel itself.
  - `HTMLWhenFieldNotShownPaging` – Switch to the Field.HTMLWhenFieldNotShown property on paging Fields
  - `HTMLWhenFieldNotShownAllFields` – Switch to the Field.HTMLWhenFieldNotShown property on all Fields.

- **LoadDefaultStyleSheetFile** (boolean) – Determines if the **~/DynamicData/Content/DESDynamicDataControls.css** style sheet file is loaded. This file contains the default style sheet classes used throughout this control.

  When `true`, the default file is loaded.

  It defaults to `true`.

- **DisableButtonsOnSubmit** (Boolean) – When `true`, the buttons will be disabled after the page submits. This has no effect when using the **QueryStringField** property.

  Updates the **DisableOnSubmit** property on the various button controls used by the subclass, so long as they are using DES's buttons. If a hyperlink is used, this has no effect.

  Requires a license for the Peter's Interactive Pages module. If not available, this property is ignored.

  When `true`, the buttons will disable on submit.

  It defaults to `false`.

- **ButtonsMayMoveOnClick** (Boolean) – If the button requires an extra click to submit the page, its because it jumped as the user clicks on it. Set this to `true` to avoid that extra click.

  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 the ValidationSummary, the button may jump. This happens before the button's onclick event, preventing that onclick event to run because the mouse button is no longer on the button.

  Updates the **MayMoveOnClick** property on the various button controls used by the subclass, so long as they are using DES's buttons. If a hyperlink is used, this has no effect.

  When `true`, the feature is enabled.

  It defaults to `false`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties on Field Classes

All Field classes are subclassed from `PeterBlum.DES.DynamicData.BaseControlPanelField` which itself is subclassed from `System.Web.UI.WebControls.DataPagerField`. Most properties on the Field classes are found in BaseControlPanel. These properties are discussed in the first section. The remaining sections are for properties on specific Field classes.

Click on any of these topics to jump to them:

- Properties common to all Field Classes
- Properties on NextPreviousPagerField, NextPagerField, and PreviousPagerField
- Properties on NumericPagerField
- Properties on all ExtendedNumericPagerFields
- Properties on InfoPagerField
- Properties on DropDownPageSizesField and LinksPageSizesField
- Properties on DropDownSorterField and LinksSorterField
- Properties on CommandButtonField and NewCommandButtonField
- Properties on DynamicButtonsField
- Properties on TemplateControlPanelField

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties common to all Field Classes

These properties are declared on `PeterBlum.DES.DynamicData.BaseControlPanelField` and `System.Web.UI.WebControls.DataPagerField` classes.

Click on any of these topics to jump to them:

- Container Properties
- Appearance Properties
- Behavior Properties
- Event Handlers

Container Properties

- **ContainerTag** (enum PeterBlum.DES.DynamicData.DynamicControlPanelContainerTag) – This Field is optionally enclosed in an HTML tag. This property determines the tag type. When used, the **ContainerTagAttributes** property provides style and other attributes to the tag.

  The enumerated type DynamicControlPanelContainerTag has these values:

  - `None` – Do not enclose the Fields in an HTML container.
  - `Auto` – Determines the tag based on the DynamicControlPanel's **ContainerTag**. If **DynamicControlPanel.ContainerTag** is `<tr>`, it uses `<td>`. If **DynamicControlPanel.ContainerTag** defines a `<ul>` or `<ol>` tag, it will use the `<li>` tag.

    For all others, it uses `<div>`.
  - `Span` – The `<span>` tag.
  - `Div` – The `<div>` tag.
  - `TableRow` – The `<tr>` tag.
  - `TableRowWithOneCell` – A `<tr>` tag and one `<td>` cell with the attribute `colspan="9999"`.
  - `TableCell` – The `<td>` tag.
  - `TableHeaderCell` – The `<th>` tag.
  - `Fieldset` – The `<fieldset>` tag.
  - `UL` – The `<ul>` tag
  - `OL` – The `<ol>` tag
  - `LI` – The `<li>` tag

  It defaults to `DynamicControlPanelContainerTag.Auto`.

- **ContainerTagAttributes** (PeterBlum.DES.DynamicData.HTMLTagAttributes) – The `style=`, `class=`, and other attributes applied to the containing HTML tag that is specified in the **ContainerTag** property. It is not used when **ContainerTag** = `None`.

  When **ContainerTag** = `TableRowWithOneCell`, the style is applied to the `<tr>` tag.

  It sets **CssClass** to "`ControlPanelField_ContainerTag`" by default, which is found in **~/DynamicData/Content/DESDynamicDataControls.css**.

  This property is defined in the ASP.NET markup like this:

```
<desDD:FieldClass ContainerTag="tag name" >
  <ContainerTagAttributes CssClass="classname" Style="css properties"
    AttributeName="AttributeValue" />
</desDD:FieldClass>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Setting the Style Sheet Class

Define a style sheet class to describe the borders, font, and background of the container around the Fields. Assign its name to the **CssClass** property on **ContainerTagAttributes**.

Setting Style Properties

Use the **Style** property on the **ContainerTagAttributes** property to declare a list of cascading style sheet styles in the CSS syntax. They are assigned to the HTML tag within the `style=` attribute.

The CSS syntax is this pattern:

```
propertyname:value; propertyname2:value2;
```

For example:

```
vertical-align:bottom; white-space:nowrap
```

The CSS property name and value must be identical to what the browser expects or they will not be used.

You can locate the CSS properties here: https://developer.mozilla.org/en/CSS_Reference

When entering the **Style** property in ASP.NET markup, Intellisense is available to assist you in picking style names and defining their values.

When working programmatically, the **Style** property can be assigned in two ways:

To add a single string containing the CSS syntax that replaces the current value of Style, do this:

```
FieldObject.ContainerTagAttributes.Style.Value = "propertyname:value;
propertyname2:value2;"
```

To add a single property, do this:

```
FieldObject.ContainerTagAttributes.Style.Add("propertyname", "value")
```

Smart Styles

When you define any of these style names, you may get the following behaviors:

`text-align` – When the HTML tag is `<td>` or `<th>`, this will be switched to an `align=` attribute if the browser does not support text-align on table cells.

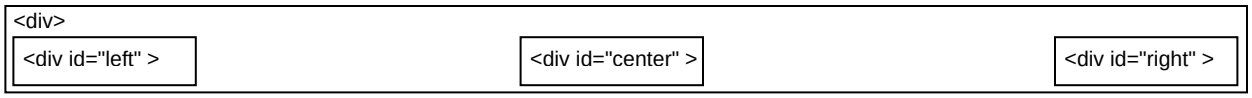
`vertical-align` – When the HTML tag is `<td>` or `<th>`, this will be switched to a `valign=` attribute if the browser does not support vertical-align on table cells.

`display` – If following values are not supported by the browser, the display property is omitted: inline-block, table, table-row, table-cell, list-item, and inline-table.

Style Suggestions

- **Horizontal Alignment** – The `text-align` CSS attribute is recommended. However, it does not work when centering "block" type tags, like and <p> within another tag. Instead use `margin-left:auto;margin-right:auto;` around the inner block as described here: http://www.w3.org/Style/Examples/007/center.

  Suppose that you want to position three tags within another, where the first is left justified, the second is centered, and the third is right justified.

  Use the float style to establish left and right positioning on the tags. Use the `text-align` style on the container div to center the rest. Recommendation: Position the HTML elements that use float before any that do not.

```
<div style="text-align" >
   <div id="left" style="float:left;" ></div>
   <div id="right" style="float:right;" ></div>
   <div id="center" ></div>
</div>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **Vertical Alignment** – The `vertical-align` CSS attribute only works on table cells and elements with display:table-cell. (Further warning: `display:table-cell` is not supported on many browsers.) Vertical alignment without tables can be tricky.

  Suggestion: When vertical alignment is needed, change to `<table>` and use the `vertical-align` style on the `<td>` tag.

  If tables are not suitable for you, here are a few articles to assist you:

  http://www.jakpsatweb.cz/css/css-vertical-center-solution.html

  http://phrogz.net/CSS/vertical-align/index.html

  Be aware that browsers differ in their support for style features like `display: table-cell` and `display:inline-block`. Even browsers that support `table-cell` may implement it differently. See http://www.quirksmode.org/css/display.html.

- **Preventing word wrap** – To keep text on the same line, use the `white-space:nowrap` style. It does not always work though.

Setting other HTML tag attributes

When the **ContainerTag's** HTML tag needs other attributes than `style=` and `class=`, enter them as follows.

- Using ASP.NET Markup, enter them as properties in the `<ContainerTagAttributes>` node like this:

```
<desDD:FieldClass ContainerTag="tag name" >
    <ContainerTagAttributes CssClass="FieldClass" cellspacing="1" />
</desDD:FieldClass>
```

  The property name should be exactly as you want to output into the HTML tag. If you are using XHTML syntax, remember to keep the property name lowercase.

- When programmatically adding attributes, call the SetAttribute() method on the ContainerTagAttributes property like this:

```
FieldObject.ContayinerTagAttributes.SetAttribute("property name", "value")
```

  For example:

```
FieldObject.ContainerTagAttributes.SetAttribute("cellspacing", "1")
```

- **ContainerTagTitle** (string) – Provides a tooltip or `<legend>` tag (when using `<fieldset>`) for the **ContainerTag**.

  For all **ContainerTags**, it supplies the tooltip (`title=` attribute). For fieldset, it also supplies the legend. It is not used when **ContainerTag** = `None`.

  It defaults to "".

- **ContainerTagTitleLookupID** (string) – Gets the value for **ContainerTagTitle** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **ContainerTagTitle** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **ContainerTagUsage** (enum PeterBlum.DES.DynamicData.ContainerTagUsage) – Determines how the **ContainerTag** property is applied to enclose the Field. An HTML tag has both opening and closing parts. This selects which of those is output, allowing multiple DataPagerFields to be enclosed within a single tag.

  An example of using this: PreviousPagerField, InfoPagerField and NextPagerField side-by-side, all affected by the same float:right style:

```
<desDD:PreviousPagerField ContainerTag="Div" ContainerTagUsage="OpeningTag" >
   <ContainerTagAttributes Style="float:right;" />
</desDD:PreviousPagerField>
<desDD:InfoPagerField ContainerTag="None" />
<desDD:NextPagerField ContainerTag="Div" ContainerTagUsage="ClosingTag"/>
```

  Set the **ContainerTag** to the same value on the first and last Field in the group. Set **ContainerTag** to `None` on those inbetween.

  The enumerated type ContainerTagUsage has these values:

  - `Full` - Fully enclose with opening and closing tags. This is the default.
  - `OpeningTag` - The opening tag only, preceding the Field.
  - `ClosingTag` - The closing tag only, following the Field.

- **HTMLBefore** (string) – HTML that appears before the field's contents. If the **ContainerTag** is used, this HTML appears before the opening container tag.

  The overall output with **ContainerTag**, **HTMLBefore**, **HTMLAfter**, and content taken into account is this:

  [HTMLBefore][OpeningContainerTag]*content*[ClosingContainerTag][HTMLAfter].

  This text is added even when **HTMLWhenFieldNotShown** is used and there is no **ContainerTag**. It is ignored if the **ContainerTag** is any of these tags: `TableRow`, `TableHeaderCell`, `TableCell`, or `LI`.

  For Fields that have a **Layout** property and you want the HTML inside the container tag, add it first to the **Layout** property instead of using **HTMLBefore**.

  It defaults to "".

- **HTMLBeforeLookupID** (string) – Gets the value for **HTMLBefore** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **HTMLBefore** 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 "".

- **HTMLAfter** (string) – HTML that appears after the field's contents. If the **ContainerTag** is used, this HTML appears after the closing container tag.

  The overall output with **ContainerTag**, **HTMLBefore**, **HTMLAfter**, and content taken into account is this:

  [HTMLBefore][OpeningContainerTag]*content*[ClosingContainerTag][HTMLAfter].

  This text is added even when **HTMLWhenFieldNotShown** is used and there is no **ContainerTag**. It is ignored if the **ContainerTag** is any of these tags: `TableRow`, `TableHeaderCell`, `TableCell`, or `LI`.

  For Fields that have a **Layout** property and you want the HTML inside the container tag, add it last to the **Layout** property instead of using **HTMLAfter**.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **HTMLAfterLookupID** (string) – Gets the value for **HTMLAfter** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **HTMLAfter** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Appearance Properties

- **HTMLWhenFieldNotShown** (string) – When the Field object has nothing to show, it will show this HTML in its place.

  A container tag around the field is only applied around this field's contents when **HTMLWhenFieldNotShownUsesContainerTag** is `true`.

  It defaults to "".

- **HTMLWhenFieldNotShownLookupID** (string) – Gets the value for **HTMLWhenFieldNotShown** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **HTMLWhenFieldNotShown** 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 "".

- **HTMLWhenFieldNotShownUsesContainerTag** (Boolean) – Determines if the HTML from **HTMLWhenFieldNotShown** is enclosed by a container tag or not. The container tag is determined by the **ContainerTag** and **ContainerTagAttributes** properties.

  It defaults to `false`.

Behavior Properties

- **HideWithOnePage** (Boolean) – Determines if the standard output is used when there is only one Page to show. If `true`, the **HTMLWhenFieldNotShown** is used. When `false`, the standard output is used.

  It defaults to `false`.

- **Visible** (Boolean) – When `false`, this Field is not rendered. It defaults to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Event Handlers

- **BeforeCreatingControls -** Event handler fired before any controls are created. Use it to modify properties of the Field, which is available in the **Field** property of the *Args* parameter. Details are shown below.
- **AfterCreatingControls** – Event handler fired after any controls are created. Use it to customize the controls that were created. They are in Container of the Args parameter. Use Container.FindControl("*ID*") to retrieve a control.

Method Declaration for BeforeCreatingControls and AfterCreatingControls

[C#]

```
void MethodName(object pSender,
  PeterBlum.DES.DynamicData.ControlPanelFieldEventArgs pArgs)
```

[VB]

```
Sub MethodName(ByVal pSender As Object,
  ByVal pArgs As PeterBlum.DES.DynamicData.ControlPanelFieldEventArgs)
```

Parameters

pSender

The Field class. This value is also available and correctly typecast in pArgs.Field.

pArgs

Properties and methods to assist you. They are described below.

ControlPanelFieldEventArgs Class

This class is passed as the argument parameter of the two event handlers described here. Here are its parameters:

- **Container** (System.Web.UI.Control) – Gets a reference to the control that hosts the controls created by the Field class. Use it with the **AfterCreatingControls** event to retrieve controls that you want to modify. Call FindControl("ID") to return a control by its ID. You can modify the Controls collection of Container to add or remove controls as you need.
- **Field** (PeterBlum.DES.DynamicData.BaseControlPanelField) – Gets a reference to the Field class. This is often modified in the **BeforeCreatingControls** event to customize how it generates controls. Typecast to the appropriate type or use any of these properties: **NextPreviousPagerField**, **InfoPagerField**, **PageSizesField**, **SorterField**, and **TemplateControlPanelField**.
- **DynamicControlPanel** (PeterBlum.DES.DynamicData.DynamicControlPanel) – Gets a reference to the DynamicControlPanel control that owns the Field.
- **PagingState** (PeterBlum.DES.DynamicData.BaseControlPanelField.PagingState) – Gets a reference to a class that describes the current paging state of the DataBound control. It has these properties and methods to assist you:
    - **StartRowIndex** (integer) – The index of the first record shown.
    - **MaximumRows** (integer) – The maximum number of rows that can be shown.
    - **TotalRowCount** (integer) – The total number of rows shown. It may be less than the Maximum Rows on the last page shown.
    - `GetTotalPages()` – Method. Returns an integer. The Total number of pages available.
    - `GetLastPageIndex()` – Method. Returns an integer. The page index to the last page. If there are 0 pages, it returns -1.
    - `GetPrevPageIndex()` – Method. Returns an integer. The page index to the previous page from the current one. The first page is 0.
    - `GetNextPageIndex()` – Method. Returns an integer. The page index to the next page from the current one.
    - `GetCurrentPageIndex()` – Method. Returns an integer. The page index currently shown. If there are no records, it returns -1.
    - `GetLastRowIndex()` – The index to the last record itself. (StartRowIndex + TotalRowCount – 1)

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties on NextPreviousPagerField, NextPagerField, and PreviousPagerField

For an overview of these Field classes, see "Using the NextPreviousPagerField" and "Using the PreviousPagerField and NextPagerField".

The `PeterBlum.DES.DynamicData.NextPreviousPagerField` inherits from `PeterBlum.DES.DynamicData.BaseControlPanelField`. For properties from its base class, see "Properties common to all Field Classes".

The NextPageField and PreviousPageField both subclass from NextPreviousPagerField. They only differ in their default to the **Layout** property, where they omit tokens for unwanted buttons.

Click on any of these topics to jump to them:

- Properties common to all Field Classes
- Layout Pattern Properties
- Buttons – Formatting Properties
- Buttons – Content Properties
- Buttons – Special Cases Properties

Layout Pattern Properties

These properties are found in the "Layout Pattern" category of Properties window. These properties determine the positioning of the controls of this Field and HTML between them.

- **Layout** (string) – Determines the HTML and positions of controls that lay out this Field. It uses tokens to represent where the command controls appear.

Token	Description
{FIRST}	The position for First Page command button.
{PREV}	The position for Previous Page command button.
{NEXT}	The position for Next Page command button.
{LAST}	The position for Last Page command button.
{IMAGE}	The position for an optional image. It is used when the **ImageUrl** property is assigned.

  Omit any token of a command that you do not want to show.

  Do not use the left or right bracket characters as literal text. They are reserved for tokens. Instead, use `{` and `}`.

  The defaults depend on the Field class:

NextPreviousPagerField	`"{FIRST} {PREV} {NEXT} {LAST}"`
PreviousPagerField	`"{FIRST} {PREV}"`
NextPagerField	`"{NEXT} {LAST}"`

- **LayoutLookupID** (string) – Gets the value for **Layout** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **Layout** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **ImageUrl** (string) – The URL to an image file. It is used when the "{IMAGE}" token is declared in the **Layout** property, allowing you to easily add an image to layout. The image replaces the token.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  It defaults to "" .

- **ImageAltText** (string) – When the browser cannot load the Image specified by **ImageUrl**, this text is displayed. It forms the `alt=` attribute. It is always written out. It defaults to "".

- **ImageAltTextLookupID** (string) – Gets the value for **ImageAltText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **ImageAltText** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Buttons – Formatting Properties

These properties are found in the "Buttons - Formatting" category of Properties window.

- **ButtonType** (enum System.Web.UI.WebControls.ButtonType) – The type of button to use for each command:
  - `Button` - A push button
  - `Link` - A Link button. It supports a blend of text and images by using the **LinkButtonLayout** property.
  - `Image` - An Image button. Requires the **ImageUrl** property to be assigned.

  The actual contents of the button are determined by the **Text** and **ImageUrl** properties.

  It defaults to `ButtonType.Link`.

- **ButtonCssClass** (string) – The style sheet class applied to each button.

  To use automatic browser renaming of the class, start with an ! character.

  It defaults to `"NextPreviousPagerField_Buttons"` which is found in **~/DynamicData/Content/DESDynamicDataControls.css**.

- **LinkButtonLayout** (enum PeterBlum.DES.DynamicData.LinkButtonLayout) – When using **ButtonType**=`Link`, it can show either or both text and image. This determines the actual layout. When set to `Auto`, only one of the two are shown. Otherwise both are shown.

  The enumerated type LinkButtonLayout has these values:

  - `Auto` - Only one is shown. If the ImageUrl is defined, an image is shown. Otherwise the text is shown.
  - `ImageOutside` - Both are shown. On the First and Previous button, the image is to the left of the text. On the Next and Last button, the image is to the right of the text. This option uses **LinkButtonLayoutSeparator** between the two elements.
  - `ImageInside` - Both are shown. On the First and Previous button, the image is to the right of the text. On the Next and Last button, the image is to the left of the text.This option uses **LinkButtonLayoutSeparator** between the two elements.

  It defaults to `LinkButtonLayout.Auto`.

- **LinkButtonLayoutSeparator** (string) – HTML placed in between an image and textual LinkButton when **ButtonType**=`Link` and **LinkButtonLayout** indicates `ImageInside` or `ImageOutside`.

  It defaults to `" "`.

- **MultipleImages** (Boolean) – When true, you have images for pressed and mouseover effects. The **ImageUrl** points to the normal graphic. Pressed and MouseOver are files in the same folder with the "Pressed" and "MouseOver" inserted in name, just before the file extension.

  The code will detect the presence of these files. If neither are found, it is not used. If one is found, it is used and the other is not. So it is safe to set this to `true` all of the time, except for the extra overhead of time used.

  The **PeterBlum.DES.Globals.Page.EnableButtonImageEffects** property can override this behavior.

  Note: This feature requires a license covering the Peter's Interactive Pages module.

  When false, it does not look for multiple images. However, the **ImageUrl** can override this by having a pipe delimited list of URLs in this format: `Normal|Pressed|MouseOver`.

  When `true`, it looks for multiple images.

  It defaults to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Buttons – Content Properties

These properties are found in the "Buttons - Content" category of Properties window.

- **FirstPageText** (string) – The label for the First Page command button.

  When using **FirstPageImageUrl**, this is used as the alternative text and tooltip.

  It defaults to `"First"`.

- **FirstPageTextLookupID** (string) – Gets the value for **FirstPageText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **FirstPageText** 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 "".

- **FirstPageImageUrl** (string) – The optional image for the First Page command button. The URL to an image that is shown when the **ButtonType** is Image or Link or the DynamicControlPanel uses the QueryString feature.

  If this is a path to a file on your local hard disk ("/image.gif" instead of http://www.somewhereelse.com/image.gif") you can also supply a second image file to use when the Button is pressed. Simply create a file in the same path with the word "Pressed" added before the file extension.

  For example, if this is "/Images/FirstButton.gif", also have "/Images/FirstButtonPressed.gif" Similarly, a mouse over is created by having an image file with the same path and "MouseOver" added before the extension.

  Supports the use of the tilde (~) as the first character to be replaced by the virtual path to the web application.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  Supports a pipe delimited list of URLs for normal|pressed|mouseover when the **MultipleImages** property is `false`. *Note: This feature requires a license covering the Peter's Interactive Pages module.*

  DES supplies a number of image files you could use:

  {APPEARANCE}/Shared/PgFirst.GIF (This is the same as the files supplied in ~/DynamicData/Contents/Images except it has corresponding Pressed and MouseOver image files.)

  {APPEARANCE}/Date And Time/LeftCmd2*.GIF (Note the * for multiple matches)

  It defaults to `"{APPEARANCE}/Shared/PgFirst.GIF"`.

- **PreviousPageText** (string) – The label for the Previous Page command button.

  When using **PreviousPageImageUrl**, this is used as the alternative text and tooltip.

  It defaults to `"Previous"`.

- **PreviousPageTextLookupID** (string) – Gets the value for **PreviousPageText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **PreviousPageText** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **PreviousPageImageUrl** (string) – The optional image for the Previous Page command button. The URL to an image that is shown when the **ButtonType** is Image or Link or the DynamicControlPanel uses the QueryString feature.

  If this is a path to a file on your local hard disk ("/image.gif" instead of http://www.somewhereelse.com/image.gif") you can also supply a second image file to use when the Button is pressed. Simply create a file in the same path with the word "Pressed" added before the file extension.

  For example, if this is "/Images/PreviousButton.gif", also have "/Images/PreviousButtonPressed.gif" Similarly, a mouse over is created by having an image file with the same path and "MouseOver" added before the extension.

  Supports the use of the tilde (~) as the Previous character to be replaced by the virtual path to the web application.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  Supports a pipe delimited list of URLs for normal|pressed|mouseover when the **MultipleImages** property is `false`. *Note: This feature requires a license covering the Peter's Interactive Pages module.*

  DES supplies a number of image files you could use:

  {APPEARANCE}/Shared/PgPrevious.GIF (This is the same as the files supplied in ~/DynamicData/Contents/Images except it has corresponding Pressed and MouseOver image files.)

  {APPEARANCE}/Date And Time/LeftCmd2*.GIF (Note the * for multiple matches)

  It defaults to `"{APPEARANCE}/Shared/PgPrevious.GIF"`.

- **NextPageText** (string) – The label for the Next Page command button.

  When using **NextPageImageUrl**, this is used as the alternative text and tooltip.

  It defaults to `"Next"`.

- **NextPageTextLookupID** (string) – Gets the value for **NextPageText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **NextPageText** 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 "".

- **NextPageImageUrl** (string) – The optional image for the Next Page command button. The URL to an image that is shown when the **ButtonType** is Image or Link or the DynamicControlPanel uses the QueryString feature.

  If this is a path to a file on your local hard disk ("/image.gif" instead of http://www.somewhereelse.com/image.gif") you can also supply a second image file to use when the Button is pressed. Simply create a file in the same path with the word "Pressed" added before the file extension.

  For example, if this is "/Images/NextButton.gif", also have "/Images/NextButtonPressed.gif" Similarly, a mouse over is created by having an image file with the same path and "MouseOver" added before the extension.

  Supports the use of the tilde (~) as the Next character to be replaced by the virtual path to the web application.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  Supports a pipe delimited list of URLs for normal|pressed|mouseover when the **MultipleImages** property is `false`. *Note: This feature requires a license covering the Peter's Interactive Pages module.*

  DES supplies a number of image files you could use:

  {APPEARANCE}/Shared/PgNext.GIF (This is the same as the files supplied in ~/DynamicData/Contents/Images except it has corresponding Pressed and MouseOver image files.)

  {APPEARANCE}/Date And Time/LeftCmd2*.GIF (Note the * for multiple matches)

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

It defaults to `"{APPEARANCE}/Shared/PgNext.GIF"`.

- **LastPageText** (string) – The label for the Last Page command button.

  When using **LastPageImageUrl**, this is used as the alternative text and tooltip.

  It defaults to `"Last"`.

- **LastPageTextLookupID** (string) – Gets the value for **LastPageText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **LastPageText** 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 "".

- **LastPageImageUrl** (string) – The optional image for the Last Page command button. The URL to an image that is shown when the **ButtonType** is Image or Link or the DynamicControlPanel uses the QueryString feature.

  If this is a path to a file on your local hard disk ("/image.gif" instead of http://www.somewhereelse.com/image.gif") you can also supply a second image file to use when the Button is pressed. Simply create a file in the same path with the word "Pressed" added before the file extension.

  For example, if this is "/Images/LastButton.gif", also have "/Images/LastButtonPressed.gif" Similarly, a mouse over is created by having an image file with the same path and "MouseOver" added before the extension.

  Supports the use of the tilde (~) as the Last character to be replaced by the virtual path to the web application.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  Supports a pipe delimited list of URLs for normal|pressed|mouseover when the **MultipleImages** property is `false`. *Note: This feature requires a license covering the Peter's Interactive Pages module.*

  DES supplies a number of image files you could use:

  {APPEARANCE}/Shared/PgLast.GIF (This is the same as the files supplied in ~/DynamicData/Contents/Images except it has corresponding Pressed and MouseOver image files.)

  {APPEARANCE}/Date And Time/LeftCmd2*.GIF (Note the * for multiple matches)

  It defaults to `"{APPEARANCE}/Shared/PgLast.GIF"`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Buttons – Special Cases Properties

These properties are found in the "Buttons – Special Cases" category of Properties window.

- **RenderDisabledButtonsAsLabels** (Boolean) – Determines if disabled command buttons are shown as Label controls instead of disabled buttons.

  Buttons are automatically disabled as the first and last page are shown. If this property is `false`, the command button will still appear disabled, using **WhenDisabledCssClass** and from by the disabled attribute on the HTML tag.

  When `true`, a Label is used. When `false`, a button of the **ButtonType** is used and shown with a disabled style from **WhenDisabledCssClass** and from setting the disabled attribute on the HTML tag.

  It defaults to `false`.

- **WhenDisabledCssClass** (string) – The style sheet class applied to each Button when disabled.

  To use automatic browser renaming of the class, start with an ! character.

  When "", opacity styles are applied automatically.

  It defaults to "". Consider setting it to `"NextPreviousPagerField_WhenDisabled"` which is found in **~/DynamicData/Content/DESDynamicDataControls.css**.

- **HideFirstLastNearLimits** (Boolean) – Hides the First Page and Last Page command buttons when the page index is very near the outer limits of pages. The First Page button is hidden when the page is **HideLimit** away from the first page. The Last Page button is hidden when the page is **HideLimit** away from the last page.

  When hidden, the **HiddenFirstHTML** and **HiddenLastHTML** properties are used in place of the actual button control. They replace the "{FIRST}" and "{LAST}" tokens in the **Layout** property.

  When `true`, hide the commands when near the limits.

  When `false`, always show the commands.

  It defaults to `false`.

- **HideLimit** (integer) – Determines the limits for the **HideFirstLastNearLimits** feature. It is the number of pages offset from the outer limits (first or last page) to hide the associated control.

  A value of 0 indicates hide on the actual first or last page.

  A value of 1 indicates hide on the second and second to last page, plus the first and last pages.

  Higher values are permitted.

  It defaults to 1.

- **HiddenFirstHTML** (string) – When the First Page command button is hidden by the **HideFirstLastNearLimits** property, this HTML replaces the "{FIRST}" token in the **Layout** property.

  It defaults to "".

- **HiddenFirstHTMLLookupID** (string) – Gets the value for **HiddenFirstHTML**through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **HiddenFirstHTML**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 "".

- **HiddenLastHTML** (string) – When the Last Page command button is hidden by the **HideFirstLastNearLimits** property, this HTML replaces the "{LAST}" token in the **Layout** property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **HiddenLastHTMLLookupID** (string) – Gets the value for **HiddenLastHTML**through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **HiddenLastHTML**will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties on NumericPagerField

For an overview of the Field class, see "Using the NumericPagerField".

The `PeterBlum.DES.DynamicData.NumericPagerField` inherits from `PeterBlum.DES.DynamicData.NextPreviousPagerField`. NextPreviousPagerField provides the buttons for First Page, Next Page, Previous Page, and Last Page.

Click on any of these topics to jump to them:

- Properties common to all Field Classes
- Layout Pattern Properties
- Buttons – Formatting Properties
- Sequential Numbering Properties
- Buttons – Content Properties
- Buttons – Special Cases Properties

Layout Pattern Properties

These properties are found in the "Layout Pattern" category of Properties window. These properties determine the positioning of the controls of this Field and HTML between them.

- **Layout** (string) – Determines the HTML and positions of controls that lay out this Field. It uses tokens to represent where the command controls appear.

Token	Description
{NUMBERS}	The position for the numeric sequence of page numbers.
{FIRST}	The position for First Page command button.
{PREV}	The position for Previous Page command button.
{NEXT}	The position for Next Page command button.
{LAST}	The position for Last Page command button.
{IMAGE}	The position for an optional image. It is used when the **ImageUrl** property is assigned.

  Omit any token of a command that you do not want to show. However if the {NUMBERS} token is missing, the numeric sequence of page numbers will still be added as the last element of the Layout. If Layout is unassigned, it is the same as "{NUMBERS}".

  Do not use the left or right bracket characters as literal text. They are reserved for tokens. Instead, use `{` and `}`.

  It defaults to `"{PREV} {NUMBERS} {NEXT}"`.

  Here are some other popular values for **Layout**.

Only numbers	`"{NUMBERS}"` or unassigned
Page commands on the left	`"{PREV} {NEXT} {NUMBERS}"`
All Page commands	`"{FIRST} {PREV} {NUMBERS} {NEXT} {LAST}"`

- **LayoutLookupID** (string) – Gets the value for **Layout** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **Layout** 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.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

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 "".

- **ImageUrl** (string) – The URL to an image file. It is used when the "{IMAGE}" token is declared in the **Layout** property, allowing you to easily add an image to layout. The image replaces the token.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  It defaults to "" .

- **ImageAltText** (string) – When the browser cannot load the Image specified by **ImageUrl**, this text is displayed. It forms the `alt=` attribute. It is always written out. It defaults to "".

- **ImageAltTextLookupID** (string) – Gets the value for **ImageAltText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **ImageAltText** 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 "".

- **PageNumberSeparator** (string) – HTML added between two page numbers that are in the numeric sequential range or in the extended range (from any of the ExtendedNumericPagerFields). Not used to indicate a gap.

  It defaults to `" "`.

- **GapSeparator** (string) – HTML added between two page numbers that are separated by a gap. A gap is when two page numbers appear side-by-side but have more than one page number between them. For example: 1 2 3 *gap* 10 *gap* 20.

  It defaults to `"…"` which is an ellipsis.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Buttons – Formatting Properties

These properties are found in the "Buttons - Formatting" category of Properties window.

See also "Buttons – Formatting Properties" on the base class, NextPreviousPagerField.

- **NextPrevButtonType** (enum System.Web.UI.WebControls.ButtonType) – The type of button to use for each command:
  - `Button` - A push button
  - `Link` - A Link button. It supports a blend of text and images by using the **LinkButtonLayout** property.
  - `Image` - An Image button. Requires the **ImageUrl** property to be assigned.

  The actual contents of the button are determined by the **Text** and **ImageUrl** properties.

  It defaults to `ButtonType.Link`.

- **NumericButtonType** (enum System.Web.UI.WebControls.ButtonType) – The type of button to use for each page number.

  `ButtonType.Image` is NOT supported. If set, an exception occurs.

  It defaults to `ButtonType.Link`.

- **NextPreviousButtonCssClass** (string) – The style sheet class applied to these command buttons: First Page, Previous Page, Next Page, and Last Page.

  To use automatic browser renaming of the class, start with an ! character.

  It defaults to `"NextPreviousPagerField_Buttons"` which is found in **~/DynamicData/Content/DESDynamicDataControls.css**.

- **NumericButtonCssClass** (string) – The style sheet class applied to each page number button, except the current page (which is not a button).

  To use automatic browser renaming of the class, start with an ! character.

  It defaults to `"NumericPagerField_Buttons"` which is found in **~/DynamicData/Content/DESDynamicDataControls.css**.

- **CurrentPageLabelCssClass** (string) – The style sheet class applied to the current page number, which is a Label, not a button.

  To use automatic browser renaming of the class, start with an ! character.

  It defaults to `"NumericPagerField_CurrentPageNumber"` which is found in **~/DynamicData/Content/DESDynamicDataControls.css**.

- **LinkButtonLayout** (enum PeterBlum.DES.DynamicData.LinkButtonLayout) – When using **NextPrevButtonType**=`Link`, it can show either or both text and image. This determines the actual layout. When set to `Auto`, only one of the two are shown. Otherwise both are shown.

  The enumerated type LinkButtonLayout has these values:

  - `Auto` - Only one is shown. If the ImageUrl is defined, an image is shown. Otherwise the text is shown.
  - `ImageOutside` - Both are shown. On the First and Previous button, the image is to the left of the text. On the Next and Last button, the image is to the right of the text. This option uses **LinkButtonLayoutSeparator** between the two elements.
  - `ImageInside` - Both are shown. On the First and Previous button, the image is to the right of the text. On the Next and Last button, the image is to the left of the text.This option uses **LinkButtonLayoutSeparator** between the two elements.

  It defaults to `LinkButtonLayout.Auto`.

- **LinkButtonLayoutSeparator** (string) – HTML placed in between an image and textual LinkButton when **NextPrevButtonType** =`Link` and **LinkButtonLayout** indicates `ImageInside` or `ImageOutside`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

It defaults to `" "`.

- **MultipleImages** (Boolean) – When true, you have images for pressed and mouseover effects. The **ImageUrl** points to the normal graphic. Pressed and MouseOver are files in the same folder with the "Pressed" and "MouseOver" inserted in name, just before the file extension.

  The code will detect the presence of these files. If neither are found, it is not used. If one is found, it is used and the other is not. So it is safe to set this to `true` all of the time, except for the extra overhead of time used.

  The **PeterBlum.DES.Globals.Page.EnableButtonImageEffects** property can override this behavior.

  Note: This feature requires a license covering the Peter's Interactive Pages module.

  When false, it does not look for multiple images. However, the **ImageUrl** can override this by having a pipe delimited list of URLs in this format: `Normal|Pressed|MouseOver`.

  When `true`, it looks for multiple images.

  It defaults to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Sequential Numbering Properties

These properties are found in the "Sequential Numbering" category of Properties window.

- **MaxPageNumbersShown** (integer) – The maximum number of page numbers shown in the sequential series of page numbers that enclose the current page number.

  A smaller number may be shown if there are fewer pages to show. An odd number is recommended as the current page number will have and even number of page numbers on each side.

  It defaults to 5.

- **AlwaysShowFirstLastPageNumbers** (Boolean) – Determines if page numbers 1 and the last page are always shown. They are in effect same as the First Page and Last Page commands. If the **Layout** property contains "{FIRST}" and "{LAST}" tokens, this does not apply.

  When used, the sequential and extended number ranges are enclosed in the first and last page number, both of which are links to select that page. If numbers inside do not immediately follow the limits, the **GapSeparator** is added.

  When `true` and **Layout** does not contain the "{FIRST}" token, this feature is enabled. Otherwise, it is disabled.

  It defaults to `false`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Buttons – Content Properties

These properties apply to these command buttons: First Page, Previous Page, Next Page, and Last Page.

These properties are found in the "Buttons - Content" category of Properties window.

- **FirstPageText** (string) – The label for the First Page command button.

  When using **FirstPageImageUrl**, this is used as the alternative text and tooltip.

  It defaults to `"First"`.

- **FirstPageTextLookupID** (string) – Gets the value for **FirstPageText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **FirstPageText** 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 "".

- **FirstPageImageUrl** (string) – The optional image for the First Page command button. The URL to an image that is shown when the **ButtonType** is Image or Link or the DynamicControlPanel uses the QueryString feature.

  If this is a path to a file on your local hard disk ("/image.gif" instead of http://www.somewhereelse.com/image.gif") you can also supply a second image file to use when the Button is pressed. Simply create a file in the same path with the word "Pressed" added before the file extension.

  For example, if this is "/Images/FirstButton.gif", also have "/Images/FirstButtonPressed.gif" Similarly, a mouse over is created by having an image file with the same path and "MouseOver" added before the extension.

  Supports the use of the tilde (~) as the first character to be replaced by the virtual path to the web application.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  Supports a pipe delimited list of URLs for normal|pressed|mouseover when the **MultipleImages** property is `false`. *Note: This feature requires a license covering the Peter's Interactive Pages module.*

  DES supplies a number of image files you could use:

  {APPEARANCE}/Shared/PgFirst.GIF (This is the same as the files supplied in ~/DynamicData/Contents/Images except it has corresponding Pressed and MouseOver image files.)

  {APPEARANCE}/Date And Time/LeftCmd2*.GIF (Note the * for multiple matches)

  It defaults to `"{APPEARANCE}/Shared/PgFirst.GIF"`.

- **PreviousPageText** (string) – The label for the Previous Page command button.

  When using **PreviousPageImageUrl**, this is used as the alternative text and tooltip.

  It defaults to `"Previous"`.

- **PreviousPageTextLookupID** (string) – Gets the value for **PreviousPageText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **PreviousPageText** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **PreviousPageImageUrl** (string) – The optional image for the Previous Page command button. The URL to an image that is shown when the **ButtonType** is Image or Link or the DynamicControlPanel uses the QueryString feature.

  If this is a path to a file on your local hard disk ("/image.gif" instead of http://www.somewhereelse.com/image.gif") you can also supply a second image file to use when the Button is pressed. Simply create a file in the same path with the word "Pressed" added before the file extension.

  For example, if this is "/Images/PreviousButton.gif", also have "/Images/PreviousButtonPressed.gif" Similarly, a mouse over is created by having an image file with the same path and "MouseOver" added before the extension.

  Supports the use of the tilde (~) as the Previous character to be replaced by the virtual path to the web application.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  Supports a pipe delimited list of URLs for normal|pressed|mouseover when the **MultipleImages** property is `false`. *Note: This feature requires a license covering the Peter's Interactive Pages module.*

  DES supplies a number of image files you could use:

  {APPEARANCE}/Shared/PgPrevious.GIF (This is the same as the files supplied in ~/DynamicData/Contents/Images except it has corresponding Pressed and MouseOver image files.)

  {APPEARANCE}/Date And Time/LeftCmd2*.GIF (Note the * for multiple matches)

  It defaults to `"{APPEARANCE}/Shared/PgPrevious.GIF"`.

- **NextPageText** (string) – The label for the Next Page command button.

  When using **NextPageImageUrl**, this is used as the alternative text and tooltip.

  It defaults to `"Next"`.

- **NextPageTextLookupID** (string) – Gets the value for **NextPageText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **NextPageText** 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 "".

- **NextPageImageUrl** (string) – The optional image for the Next Page command button. The URL to an image that is shown when the **ButtonType** is Image or Link or the DynamicControlPanel uses the QueryString feature.

  If this is a path to a file on your local hard disk ("/image.gif" instead of http://www.somewhereelse.com/image.gif") you can also supply a second image file to use when the Button is pressed. Simply create a file in the same path with the word "Pressed" added before the file extension.

  For example, if this is "/Images/NextButton.gif", also have "/Images/NextButtonPressed.gif" Similarly, a mouse over is created by having an image file with the same path and "MouseOver" added before the extension.

  Supports the use of the tilde (~) as the Next character to be replaced by the virtual path to the web application.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  Supports a pipe delimited list of URLs for normal|pressed|mouseover when the **MultipleImages** property is `false`. *Note: This feature requires a license covering the Peter's Interactive Pages module.*

  DES supplies a number of image files you could use:

  {APPEARANCE}/Shared/PgNext.GIF (This is the same as the files supplied in ~/DynamicData/Contents/Images except it has corresponding Pressed and MouseOver image files.)

  {APPEARANCE}/Date And Time/LeftCmd2*.GIF (Note the * for multiple matches)

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

It defaults to `"{APPEARANCE}/Shared/PgNext.GIF"`.

- **LastPageText** (string) – The label for the Last Page command button.

  When using **LastPageImageUrl**, this is used as the alternative text and tooltip.

  It defaults to `"Last"`.

- **LastPageTextLookupID** (string) – Gets the value for **LastPageText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **LastPageText** 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 "".

- **LastPageImageUrl** (string) – The optional image for the Last Page command button. The URL to an image that is shown when the **ButtonType** is Image or Link or the DynamicControlPanel uses the QueryString feature.

  If this is a path to a file on your local hard disk ("/image.gif" instead of http://www.somewhereelse.com/image.gif") you can also supply a second image file to use when the Button is pressed. Simply create a file in the same path with the word "Pressed" added before the file extension.

  For example, if this is "/Images/LastButton.gif", also have "/Images/LastButtonPressed.gif" Similarly, a mouse over is created by having an image file with the same path and "MouseOver" added before the extension.

  Supports the use of the tilde (~) as the Last character to be replaced by the virtual path to the web application.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  Supports a pipe delimited list of URLs for normal|pressed|mouseover when the **MultipleImages** property is `false`. *Note: This feature requires a license covering the Peter's Interactive Pages module.*

  DES supplies a number of image files you could use:

  {APPEARANCE}/Shared/PgLast.GIF (This is the same as the files supplied in ~/DynamicData/Contents/Images except it has corresponding Pressed and MouseOver image files.)

  {APPEARANCE}/Date And Time/LeftCmd2*.GIF (Note the * for multiple matches)

  It defaults to `"{APPEARANCE}/Shared/PgLast.GIF"`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Buttons – Special Cases Properties

These properties apply to these command buttons: First Page, Previous Page, Next Page, and Last Page.

These properties are found in the "Buttons – Special Cases" category of Properties window.

- **RenderDisabledButtonsAsLabels** (Boolean) – Determines if disabled command buttons are shown as Label controls instead of disabled buttons.

  Buttons are automatically disabled as the first and last page are shown. If this property is `false`, the command button will still appear disabled, using **WhenDisabledCssClass** and from by the disabled attribute on the HTML tag.

  When `true`, a Label is used. When `false`, a button of the **ButtonType** is used and shown with a disabled style from **WhenDisabledCssClass** and from setting the disabled attribute on the HTML tag.

  It defaults to `false`.

- **WhenDisabledCssClass** (string) – The style sheet class applied to each Button when disabled.

  To use automatic browser renaming of the class, start with an ! character.

  When "", opacity styles are applied automatically.

  It defaults to "". Consider setting it to `"NextPreviousPagerField_WhenDisabled"` which is found in **~/DynamicData/Content/DESDynamicDataControls.css**.

- **HideFirstLastNearLimits** (Boolean) – Hides the First Page and Last Page command buttons when the page index is very near the outer limits of pages. The First Page button is hidden when the page is **HideLimit** away from the first page. The Last Page button is hidden when the page is **HideLimit** away from the last page.

  When hidden, the **HiddenFirstHTML** and **HiddenLastHTML** properties are used in place of the actual button control. They replace the "{FIRST}" and "{LAST}" tokens in the **Layout** property.

  When `true`, hide the commands when near the limits.

  When `false`, always show the commands.

  It defaults to `false`.

- **HideLimit** (integer) – Determines the limits for the **HideFirstLastNearLimits** feature. It is the number of pages offset from the outer limits (first or last page) to hide the associated control.

  A value of 0 indicates hide on the actual first or last page.

  A value of 1 indicates hide on the second and second to last page, plus the first and last pages.

  Higher values are permitted.

  It defaults to 1.

- **HiddenFirstHTML** (string) – When the First Page command button is hidden by the **HideFirstLastNearLimits** property, this HTML replaces the "{FIRST}" token in the **Layout** property.

  It defaults to "".

- **HiddenFirstHTMLLookupID** (string) – Gets the value for **HiddenFirstHTML**through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **HiddenFirstHTML**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 "".

- **HiddenLastHTML** (string) – When the Last Page command button is hidden by the **HideFirstLastNearLimits** property, this HTML replaces the "{LAST}" token in the **Layout** property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **HiddenLastHTMLLookupID** (string) – Gets the value for **HiddenLastHTML**through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **HiddenLastHTML**will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties on all ExtendedNumericPagerFields

For an overview of each Field class, see "Using the IncrementExtendedNumericPagerField", "Using the SpreadExtendedNumericPagerField", and "Using the JumpByExtendedNumericPagerField"

The IncrementExtendedNumericPagerField, SpreadExtendedNumericPagerField, and JumpByExtendedNumericPagerField classes all inherit from `PeterBlum.DES.DynamicData.NumericPagerField`. The NumericPagerField provides the sequential page number range, surrounding the current page. The Extended Fields provide the *extended number range* with numbers that fill the gaps between the sequential range and first or last page.

The NumericPagerField inherits from `PeterBlum.DES.DynamicData.NextPreviousPagerField`. NextPreviousPagerField provides the buttons for First Page, Next Page, Previous Page, and Last Page.

For properties from its base classes, see "Properties common to all Field Classes".

Click on any of these topics to jump to them:

- Properties common to all Field Classes
- Extended Numbering Properties
- Sequential Numbering Properties
- Layout Pattern Properties
- Buttons – Formatting Properties
- Buttons – Content Properties
- Buttons – Special Cases Properties

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Extended Numbering Properties

These properties are found in the "Extended Numbering" category of Properties window.

- **ExtMaxPageNumbersShown** (integer) – The maximum number of page numbers shown in the extended range. *Not used by JumpByExtendedNumericPagerField.*

  A smaller number may be shown if there are fewer pages to show.

  It defaults to 5.

- **ExtMinPageNumbersShown** (integer) – The minimum number of page numbers shown in the extended range. If fewer will be generated, the extended range is replaced by the **GapSeparator**. *Not used by JumpByExtendedNumericPagerField.*

  It defaults to 1.

- **Increment** (integer) – *IncrementExtendedNumericPagerField Only.* The number of pages between each page number in this sequence.

  It defaults to 10.

- **RoundToNearest** (Boolean) – *IncrementExtendedNumericPagerField Only.* Determines if each page number is rounded to a number evenly divisible by Increment or just added to the start or end of the sequential range.

  When `true`, each page number should be rounded up using **Increment** as for numbering. For example, if **Increment** is 10, numbers must be divisble by 10. If the last sequential page number is 6, it will be followed by 10, 20, 30.

  When `false`, page numbers are offset from the **Increment** at the start or end depending on where the sequential range is shown. For example, if increment is 10 and the last sequential page number is 6, it will be followed by 16, 26, etc.

  It defaults to `false`.

- **MinimumBetweenPages** (integer) – *SpreadExtendedNumericPagerField Only.* If the calculated number between pages is less than this, fewer page numbers are shown until either they have this distance or there is nothing to shown.

  It defaults to 10.

- **JumpBy** (integer) – *JumpByExtendedNumericPagerField Only.* The number to add or subtract against the current page number.

  It defaults to 10.

- **SecondJumpBy** (integer) – *JumpByExtendedNumericPagerField Only.* A second number to add or subtract from the current page. When 0, it is not used.

  While its normal to use a higher value than **JumpBy**, it is not required. However, it will always be positioned after **JumpBy** (nearer to the limits).

  When 0, it is not used.

  It defaults to 0.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Sequential Numbering Properties

These properties are found in the "Sequential Numbering" category of Properties window.

- **MaxPageNumbersShown** (integer) – The maximum number of page numbers shown in the sequential series of page numbers that enclose the current page number.

  A smaller number may be shown if there are fewer pages to show. An odd number is recommended as the current page number will have and even number of page numbers on each side.

  It defaults to 5.

- **AlwaysShowFirstLastPageNumbers** (Boolean) – Determines if page numbers 1 and the last page are always shown. They are in effect same as the First Page and Last Page commands. If the **Layout** property contains "{FIRST}" and "{LAST}" tokens, this does not apply.

  When used, the sequential and extended number ranges are enclosed in the first and last page number, both of which are links to select that page. If numbers inside do not immediately follow the limits, the **GapSeparator** is added.

  When `true` and **Layout** does not contain the "{FIRST}" token, this feature is enabled. Otherwise, it is disabled.

  It defaults to `false`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Layout Pattern Properties

These properties are found in the "Layout Pattern" category of Properties window. These properties determine the positioning of the controls of this Field and HTML between them.

- **Layout** (string) – Determines the HTML and positions of controls that lay out this Field. It uses tokens to represent where the command controls appear.

Token	Description
{NUMBERS}	The position for the numeric sequence of page numbers.
{FIRST}	The position for First Page command button.
{PREV}	The position for Previous Page command button.
{NEXT}	The position for Next Page command button.
{LAST}	The position for Last Page command button.
{IMAGE}	The position for an optional image. It is used when the **ImageUrl** property is assigned.

  Omit any token of a command that you do not want to show. However if the {NUMBERS} token is missing, the numeric sequence of page numbers will still be added as the last element of the Layout. If Layout is unassigned, it is the same as "{NUMBERS}".

  Do not use the left or right bracket characters as literal text. They are reserved for tokens. Instead, use `{` and `}`.

  It defaults to `"{PREV} {NUMBERS} {NEXT}"`.

  Here are some other popular values for **Layout**.

Only numbers	`"{NUMBERS}"` or unassigned
Page commands on the left	`"{PREV} {NEXT} {NUMBERS}"`
All Page commands	`"{FIRST} {PREV} {NUMBERS} {NEXT} {LAST}"`

- **LayoutLookupID** (string) – Gets the value for **Layout** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **Layout** 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 "".

- **ImageUrl** (string) – The URL to an image file. It is used when the "{IMAGE}" token is declared in the **Layout** property, allowing you to easily add an image to layout. The image replaces the token.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  It defaults to "" .

- **ImageAltText** (string) – When the browser cannot load the Image specified by **ImageUrl**, this text is displayed. It forms the `alt=` attribute. It is always written out. It defaults to "".

- **ImageAltTextLookupID** (string) – Gets the value for **ImageAltText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **ImageAltText** 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.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

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 "".

- **PageNumberSeparator** (string) – *For the sequential page numbers.* HTML added between two page numbers that are in the numeric sequential range or in the extended range (from any of the ExtendedNumericPagerFields). Not used to indicate a gap.

  It defaults to `" "`.

- **GapSeparator** (string) – HTML added between two page numbers that are separated by a gap. A gap is when two page numbers appear side-by-side but have more than one page number between them. For example: 1 2 3 *gap* 10 *gap* 20.

  It defaults to `"…"` which is an ellipsis.

- **ExtPageNumberSeparator** (string) – *For the extended page numbers.* The HTML that separates each page number shown. If the "{DEFAULT}" token is specified, use **PageNumberSeparator**.

  It defaults to `"{DEFAULT}"`.

- **RangesSeparator** (string) – The HTML that separates the sequential range from the extended range. If the "{DEFAULT}" token is specified, use **ExtPageNumberSeparator**.

  It defaults to `"{DEFAULT}"`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Buttons – Formatting Properties

These properties are found in the "Buttons - Formatting" category of Properties window.

See also "Buttons – Formatting Properties" on the base class, NextPreviousPagerField.

- **NextPrevButtonType** (enum System.Web.UI.WebControls.ButtonType) – The type of button to use for each command:
  - `Button` - A push button
  - `Link` - A Link button. It supports a blend of text and images by using the **LinkButtonLayout** property.
  - `Image` - An Image button. Requires the **ImageUrl** property to be assigned.

  The actual contents of the button are determined by the **Text** and **ImageUrl** properties.

  It defaults to `ButtonType.Link`.

- **NumericButtonType** (enum System.Web.UI.WebControls.ButtonType) – The type of button to use for each page number. *Applies to both sequential and extended ranges.*

  `ButtonType.Image` is NOT supported. If set, an exception occurs.

  It defaults to `ButtonType.Link`.

- **NextPreviousButtonCssClass** (string) – The style sheet class applied to these command buttons: First Page, Previous Page, Next Page, and Last Page.

  To use automatic browser renaming of the class, start with an ! character.

  It defaults to `"NextPreviousPagerField_Buttons"` which is found in **~/DynamicData/Content/DESDynamicDataControls.css**.

- **NumericButtonCssClass** (string) – *For the sequential page numbers.* The style sheet class applied to each page number button, except the current page (which is not a button).

  To use automatic browser renaming of the class, start with an ! character.

  It defaults to `"NumericPagerField_Buttons"` which is found in **~/DynamicData/Content/DESDynamicDataControls.css**.

- **ExtNumericButtonCssClass** (string) – *For the extended page numbers.* The style sheet class applied to each page number button in the extended range.

  To use automatic browser renaming of the class, start with an ! character.

  When set to "{DEFAULT}", it uses the value from the **NumericButtonCssClass** property.

  It defaults to "`{DEFAULT}`" which uses `"NumericPagerField_Buttons"` from **~/DynamicData/Content/DESDynamicDataControls.css**.

- **CurrentPageLabelCssClass** (string) – The style sheet class applied to the current page number, which is a Label, not a button.

  To use automatic browser renaming of the class, start with an ! character.

  It defaults to `"NumericPagerField_CurrentPageNumber"` which is found in **~/DynamicData/Content/DESDynamicDataControls.css**.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **LinkButtonLayout** (enum PeterBlum.DES.DynamicData.LinkButtonLayout) – When using **NextPrevButtonType**=`Link`, it can show either or both text and image. This determines the actual layout. When set to `Auto`, only one of the two are shown. Otherwise both are shown.

  The enumerated type LinkButtonLayout has these values:

  - `Auto` - Only one is shown. If the ImageUrl is defined, an image is shown. Otherwise the text is shown.
  - `ImageOutside` - Both are shown. On the First and Previous button, the image is to the left of the text. On the Next and Last button, the image is to the right of the text. This option uses **LinkButtonLayoutSeparator** between the two elements.
  - `ImageInside` - Both are shown. On the First and Previous button, the image is to the right of the text. On the Next and Last button, the image is to the left of the text.This option uses **LinkButtonLayoutSeparator** between the two elements.

  It defaults to `LinkButtonLayout.Auto`.

- **LinkButtonLayoutSeparator** (string) – HTML placed in between an image and textual LinkButton when **NextPrevButtonType** =`Link` and **LinkButtonLayout** indicates `ImageInside` or `ImageOutside`.

  It defaults to `" "`.

- **MultipleImages** (Boolean) – When true, you have images for pressed and mouseover effects. The **ImageUrl** points to the normal graphic. Pressed and MouseOver are files in the same folder with the "Pressed" and "MouseOver" inserted in name, just before the file extension.

  The code will detect the presence of these files. If neither are found, it is not used. If one is found, it is used and the other is not. So it is safe to set this to `true` all of the time, except for the extra overhead of time used.

  The **PeterBlum.DES.Globals.Page.EnableButtonImageEffects** property can override this behavior.

  Note: This feature requires a license covering the Peter's Interactive Pages module.

  When false, it does not look for multiple images. However, the **ImageUrl** can override this by having a pipe delimited list of URLs in this format: `Normal|Pressed|MouseOver`.

  When `true`, it looks for multiple images.

  It defaults to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Buttons – Content Properties

These properties are found in the "Buttons - Content" category of Properties window.

For the JumpByExtendedNumericPagerField

- **JumpForwardText** (string) – *JumpByExtendedNumericPagerField Only.* Determines the label of the Jump Forward buttons. If blank, it uses the offset, preceded with the plus character. If you want that number to appear along with other text, use the "{JUMPBY}" or "{SIGNEDJUMPBY}" token.

  Used by both **JumpBy** and **SecondJumpBy**. If you use both and set this, you should always use one of the tokens to distinguish them from each other.

  When blank, it uses the offset number preceded by the culture specific Plus (+) character.

  The token "{JUMPBY}" is replaced by the offset number.

  The token "{SIGNEDJUMPBY}" is replaced by he offset number preceded by the culture specific Plus (+) character.

  It defaults to "".

- **JumpBackText** (string) – *JumpByExtendedNumericPagerField Only.* Determines the label of the Jump Back buttons. If blank, it uses the offset, preceded with the minus character. If you want that number to appear along with other text, use the "{JUMPBY}" or "{SIGNEDJUMPBY}" token.

  Used by both **JumpBy** and **SecondJumpBy**. If you use both and set this, you should always use one of the tokens to distinguish them from each other.

  When blank, it uses the offset number preceded by the culture specific Minus (-) character.

  The token "{JUMPBY}" is replaced by the offset number.

  The token "{SIGNEDJUMPBY}" is replaced by he offset number preceded by the culture specific Minus (-) character.

  It defaults to "".

- **JumpForwardTextLookupID** (string) – Gets the value for **JumpForwardText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **JumpForwardText** 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 "".

- **JumpBackTextLookupID** (string) – Gets the value for **JumpBackText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **JumpBackText** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

First, Previous, Next, and Last Page buttons

- **FirstPageText** (string) – The label for the First Page command button.

  When using **FirstPageImageUrl**, this is used as the alternative text and tooltip.

  It defaults to `"First"`.

- **FirstPageTextLookupID** (string) – Gets the value for **FirstPageText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **FirstPageText** 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 "".

- **FirstPageImageUrl** (string) – The optional image for the First Page command button. The URL to an image that is shown when the **ButtonType** is Image or Link or the DynamicControlPanel uses the QueryString feature.

  If this is a path to a file on your local hard disk ("/image.gif" instead of http://www.somewhereelse.com/image.gif") you can also supply a second image file to use when the Button is pressed. Simply create a file in the same path with the word "Pressed" added before the file extension.

  For example, if this is "/Images/FirstButton.gif", also have "/Images/FirstButtonPressed.gif" Similarly, a mouse over is created by having an image file with the same path and "MouseOver" added before the extension.

  Supports the use of the tilde (~) as the first character to be replaced by the virtual path to the web application.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  Supports a pipe delimited list of URLs for normal|pressed|mouseover when the **MultipleImages** property is `false`. *Note: This feature requires a license covering the Peter's Interactive Pages module.*

  DES supplies a number of image files you could use:

  {APPEARANCE}/Shared/PgFirst.GIF (This is the same as the files supplied in ~/DynamicData/Contents/Images except it has corresponding Pressed and MouseOver image files.)

  {APPEARANCE}/Date And Time/LeftCmd2*.GIF (Note the * for multiple matches)

  It defaults to `"{APPEARANCE}/Shared/PgFirst.GIF"`.

- **PreviousPageText** (string) – The label for the Previous Page command button.

  When using **PreviousPageImageUrl**, this is used as the alternative text and tooltip.

  It defaults to `"Previous"`.

- **PreviousPageTextLookupID** (string) – Gets the value for **PreviousPageText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **PreviousPageText** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **PreviousPageImageUrl** (string) – The optional image for the Previous Page command button. The URL to an image that is shown when the **ButtonType** is Image or Link or the DynamicControlPanel uses the QueryString feature.

  If this is a path to a file on your local hard disk ("/image.gif" instead of http://www.somewhereelse.com/image.gif") you can also supply a second image file to use when the Button is pressed. Simply create a file in the same path with the word "Pressed" added before the file extension.

  For example, if this is "/Images/PreviousButton.gif", also have "/Images/PreviousButtonPressed.gif" Similarly, a mouse over is created by having an image file with the same path and "MouseOver" added before the extension.

  Supports the use of the tilde (~) as the Previous character to be replaced by the virtual path to the web application.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  Supports a pipe delimited list of URLs for normal|pressed|mouseover when the **MultipleImages** property is `false`. *Note: This feature requires a license covering the Peter's Interactive Pages module.*

  DES supplies a number of image files you could use:

  {APPEARANCE}/Shared/PgPrevious.GIF (This is the same as the files supplied in ~/DynamicData/Contents/Images except it has corresponding Pressed and MouseOver image files.)

  {APPEARANCE}/Date And Time/LeftCmd2*.GIF (Note the * for multiple matches)

  It defaults to `"{APPEARANCE}/Shared/PgPrevious.GIF"`.

- **NextPageText** (string) – The label for the Next Page command button.

  When using **NextPageImageUrl**, this is used as the alternative text and tooltip.

  It defaults to `"Next"`.

- **NextPageTextLookupID** (string) – Gets the value for **NextPageText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **NextPageText** 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 "".

- **NextPageImageUrl** (string) – The optional image for the Next Page command button. The URL to an image that is shown when the **ButtonType** is Image or Link or the DynamicControlPanel uses the QueryString feature.

  If this is a path to a file on your local hard disk ("/image.gif" instead of http://www.somewhereelse.com/image.gif") you can also supply a second image file to use when the Button is pressed. Simply create a file in the same path with the word "Pressed" added before the file extension.

  For example, if this is "/Images/NextButton.gif", also have "/Images/NextButtonPressed.gif" Similarly, a mouse over is created by having an image file with the same path and "MouseOver" added before the extension.

  Supports the use of the tilde (~) as the Next character to be replaced by the virtual path to the web application.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  Supports a pipe delimited list of URLs for normal|pressed|mouseover when the **MultipleImages** property is `false`. *Note: This feature requires a license covering the Peter's Interactive Pages module.*

  DES supplies a number of image files you could use:

  {APPEARANCE}/Shared/PgNext.GIF (This is the same as the files supplied in ~/DynamicData/Contents/Images except it has corresponding Pressed and MouseOver image files.)

  {APPEARANCE}/Date And Time/LeftCmd2*.GIF (Note the * for multiple matches)

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

It defaults to `"{APPEARANCE}/Shared/PgNext.GIF"`.

- **LastPageText** (string) – The label for the Last Page command button.

  When using **LastPageImageUrl**, this is used as the alternative text and tooltip.

  It defaults to `"Last"`.

- **LastPageTextLookupID** (string) – Gets the value for **LastPageText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **LastPageText** 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 "".

- **LastPageImageUrl** (string) – The optional image for the Last Page command button. The URL to an image that is shown when the **ButtonType** is Image or Link or the DynamicControlPanel uses the QueryString feature.

  If this is a path to a file on your local hard disk ("/image.gif" instead of http://www.somewhereelse.com/image.gif") you can also supply a second image file to use when the Button is pressed. Simply create a file in the same path with the word "Pressed" added before the file extension.

  For example, if this is "/Images/LastButton.gif", also have "/Images/LastButtonPressed.gif" Similarly, a mouse over is created by having an image file with the same path and "MouseOver" added before the extension.

  Supports the use of the tilde (~) as the Last character to be replaced by the virtual path to the web application.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  Supports a pipe delimited list of URLs for normal|pressed|mouseover when the **MultipleImages** property is `false`. *Note: This feature requires a license covering the Peter's Interactive Pages module.*

  DES supplies a number of image files you could use:

  {APPEARANCE}/Shared/PgLast.GIF (This is the same as the files supplied in ~/DynamicData/Contents/Images except it has corresponding Pressed and MouseOver image files.)

  {APPEARANCE}/Date And Time/LeftCmd2*.GIF (Note the * for multiple matches)

  It defaults to `"{APPEARANCE}/Shared/PgLast.GIF"`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Buttons – Special Cases Properties

These properties apply to these command buttons: First Page, Previous Page, Next Page, and Last Page.

These properties are found in the "Buttons – Special Cases" category of Properties window.

- **RenderDisabledButtonsAsLabels** (Boolean) – Determines if disabled command buttons are shown as Label controls instead of disabled buttons.

  Buttons are automatically disabled as the first and last page are shown. If this property is `false`, the command button will still appear disabled, using **WhenDisabledCssClass** and from by the disabled attribute on the HTML tag.

  When `true`, a Label is used. When `false`, a button of the **ButtonType** is used and shown with a disabled style from **WhenDisabledCssClass** and from setting the disabled attribute on the HTML tag.

  It defaults to `false`.

- **WhenDisabledCssClass** (string) – The style sheet class applied to each Button when disabled.

  To use automatic browser renaming of the class, start with an ! character.

  When "", opacity styles are applied automatically.

  It defaults to "". Consider setting it to `"NextPreviousPagerField_WhenDisabled"` which is found in **~/DynamicData/Content/DESDynamicDataControls.css**.

- **HideFirstLastNearLimits** (Boolean) – Hides the First Page and Last Page command buttons when the page index is very near the outer limits of pages. The First Page button is hidden when the page is **HideLimit** away from the first page. The Last Page button is hidden when the page is **HideLimit** away from the last page.

  When hidden, the **HiddenFirstHTML** and **HiddenLastHTML** properties are used in place of the actual button control. They replace the "{FIRST}" and "{LAST}" tokens in the **Layout** property.

  When `true`, hide the commands when near the limits.

  When `false`, always show the commands.

  It defaults to `false`.

- **HideLimit** (integer) – Determines the limits for the **HideFirstLastNearLimits** feature. It is the number of pages offset from the outer limits (first or last page) to hide the associated control.

  A value of 0 indicates hide on the actual first or last page.

  A value of 1 indicates hide on the second and second to last page, plus the first and last pages.

  Higher values are permitted.

  It defaults to 1.

- **HiddenFirstHTML** (string) – When the First Page command button is hidden by the **HideFirstLastNearLimits** property, this HTML replaces the "{FIRST}" token in the **Layout** property.

  It defaults to "".

- **HiddenFirstHTMLLookupID** (string) – Gets the value for **HiddenFirstHTML**through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **HiddenFirstHTML**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 "".

- **HiddenLastHTML** (string) – When the Last Page command button is hidden by the **HideFirstLastNearLimits** property, this HTML replaces the "{LAST}" token in the **Layout** property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **HiddenLastHTMLLookupID** (string) – Gets the value for **HiddenLastHTML**through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **HiddenLastHTML**will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties on InfoPagerField

For an overview of this Field class, see "Using the InfoPagerField". Most of the setup is done in the **Layout** property by using its tokens.

The `PeterBlum.DES.DynamicData.InfoPagerField` inherits from `PeterBlum.DES.DynamicData.BaseControlPanelField`. For properties from its base class, see "Properties common to all Field Classes".

Click on any of these topics to jump to them:

- Properties common to all Field Classes
- Layout Pattern Properties
- Paging TextBox Properties

Layout Pattern Properties

These properties are found in the "Layout Pattern" category of Properties window. These properties determine the positioning of the controls of this Field and HTML between them.

- **Layout** (string) – Determines the HTML and positions of controls that lay out this Field. It uses tokens to represent where the command controls appear.

Token	Description
{PAGEINDEX}	The current page index where 1 is the first page. When EditablePageIndex is true, this is replaced by an IntegerTextBox whose Columns property is the number of digits in the total pages to show.
{PAGETOTAL}	The total pages that can be shown.
{PERPAGE}	The number of rows per page allowed.
{RECSTART}	The record number of the first record shown on this page.
{RECEND}	The record number of the last record shown on this page.
{RECTOTAL}	Total number of records in the query (across all pages).
{RECSHOWN}	The number of records shown on this page.
{PERCENT}	The percentage of the index / total where Index=1 is 0% and Index=Total = 100%.
{IMAGE}	The position for an optional image. It is used when the **ImageUrl** property is assigned.

  Omit any token of a command that you do not want to show.

  Do not use the left or right bracket characters as literal text. They are reserved for tokens. Instead, use `{` and `}`.

  The default is `"Page {PAGEINDEX} of {PAGETOTAL}"`.

- **LayoutLookupID** (string) – Gets the value for **Layout** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **Layout** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **ImageUrl** (string) – The URL to an image file. It is used when the "{IMAGE}" token is declared in the **Layout** property, allowing you to easily add an image to layout. The image replaces the token.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  It defaults to "" .

- **ImageAltText** (string) – When the browser cannot load the Image specified by **ImageUrl**, this text is displayed. It forms the `alt=` attribute. It is always written out. It defaults to "".

- **ImageAltTextLookupID** (string) – Gets the value for **ImageAltText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **ImageAltText** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Paging TextBox Properties

These properties setup a textbox where the {PAGEINDEX} token is shown in the Layout. It allows entry of a different page number. It provides validation of the page number range and a hint while focus is in the textbox.

These properties are found in the "Paging TextBox" category of Properties window.

- **EditablePageIndex** (Boolean) – Determines if the "{PAGEINDEX}" token shows an editable page number control or a label. When editable, an IntegerTextBox is used. It autopostbacks on a change to update the list. Its width is determined by the largest page size. (Example: If the last page is 134, the width is Columns="3".)

  Warning: The TextBox is not supported when the DynamicControlPanel uses the Querystring feature.

  To invoke an AutoPostBack, the user can hit the ENTER key as well as take any action to remove focus from the textbox.

  When `true`, the IntegerTextBox is used. When `false`, a Label is used.

  It defaults to `true`.

  The textbox automatically gets a range validator to detect pages out of range.

- **PageIndexCssClass** (string) – The style sheet class applied to the "{PAGEINDEX}" token's HTML output.

  To use automatic browser renaming of the class, start with an ! character.

  Depending on **EditablePageIndex**, this is applied to a Label or IntegerTextBox.

  It defaults to `"cDefaultPageIndexCssClass"` which is found in **~/DynamicData/Content/DESDynamicDataControls.css**.

- **RangeErrorMessage** (string) – A validator error message for when the user enters an index value that is out of range. Internally, DES will include a RangeValidator that is used to prevent AutoPostBack.

  If you set this to "", it will still validate but the ErrorFormatter will be set to **Display**=`None`, hiding it. This results in simply preventing the user from causing AutoPostBack when there isn't good data.

  It defaults to `"Enter a value between {MINIMUM} and {MAXIMUM}."`.

- **RangeErrorLookupID** (string) – Gets the value for **RangeError** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of ErrorMessages. If no match is found OR this is blank, **RangeError** 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 "".

- **ErrorFormatterSkinID** (string) – Sets the **ErrorFormatterSkinID** property on the RangeValidator to setup an ErrorFormatter. The ErrorFormatterSkinID can refer to a saved skin name. They are defined in the custom.DES.config file within the `<ErrorFormatters>` section.

  It can also provide a shorthand notation that specifies an ErrorFormatter type and any desired properties. For example, this uses the PopupErrorFormatter and sets its Display property to "Dynamic":

  ```
  "PopupView {Display='Dynamic'}"
  ```

  It defaults to "{DEFAULT}" which gets its value from the page level property **PeterBlum.DES.Globals.Page.DefaultErrorFormatterSkinID**.

  See the "ErrorFormatter Skins" section in the **Validation User's Guide** for more details.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **PageIndexHint** (string) – A hint shown on the IntegerTextBox to assist the user in entry of the PageIndex value. The formatting of the hint is determined by the **PageIndexHintPopupViewName** property.

  Note: Requires an license covering the Peter's Interactive Pages module.

  It defaults to `"Enter the page number and hit ENTER to display that page."`.

- **PageIndexHintLookupID** (string) – Gets the value for **PageIndexHint** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **PageIndexHint** 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 "".

- **PageIndexHintPopupViewName** (string) – Determines which globally defined PopupView is used for the **PageIndexHint** feature. Specify the name or use "{DEFAULT}" to select the name defined globally in the **DefaultHintPopupViewName** property of the Global Settings Editor.

  Your **custom.des.config** file contains the definitions for the PopupViews in its `<HintPopupViews>` section. Use the Global Settings Editor to edit these definitions.

  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.

  It defaults to `"{DEFAULT}"`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties on DropDownPageSizesField and LinksPageSizesField

For an overview of these Field classes, see "Using the DropDownPageSizesField and LinksPageSizesField"

The `PeterBlum.DES.DynamicData.DropDownPageSizesField` and `PeterBlum.DES.DynamicData.LinkPageSizesField` inherit from `PeterBlum.DES.DynamicData.BaseControlPanelField`. For properties from its base class, see "Properties common to all Field Classes".

Click on any of these topics to jump to them:

- Properties common to all Field Classes
- Page Sizes Properties
- Appearance Properties
- Layout Pattern Properties

Page Sizes Properties

These properties are found in the "Page Sizes" category of Properties window.

- **Sizes** (string) – A pipe (|) delimited list of the sizes offered, in the desired order. It should only include numbers and optionally the string `"ALL"`, which lets the user see all items on the same page.

  Any spaces around the numbers and commands will be trimmed before usage.

  It defaults to `"5|10|25|ALL"`.

- **AllLimit** (integer) – Establishes a maximum size for the ALL command. It is not smart to actually provide all records when the database can reflect millions. This establishes a limit to the number of records shown.

  It defaults to 200.

- **AllCommandLabel** (string) – Text shown for the ALL command. The **Sizes** property will always use the text "ALL" to reflect the ALL command. The DropDownList and Hyperlinks will replace it with this text.

  It defaults to `"All"`.

- **AllCommandLabelLookupID** (string) – Gets the value for **AllCommandLabel** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **AllCommandLabel** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Appearance Properties

These properties are found in the "Appearance" category of Properties window.

For DropDownPageSizesField

- **DropDownListCssClass** (string) – The style sheet class applied to DropDownList.

  To use automatic browser renaming of the class, start with an ! character.

  It defaults to `""`.

For LinksPageSizesField

- **LinkSeparator** (string) – The text between each hyperlink. It defaults to `" "`.
- **LinkCssClass** (string) – The style sheet class applied to each hyperlink.

  To use automatic browser renaming of the class, start with an ! character.

  It defaults to `"LinksPageSizesField_Link"` which is found in **~/DynamicData/Content/DESDynamicDataControls.css**.
- **CurrentSizeCssClass** (string) – The style sheet class applied to the current page size label.

  To use automatic browser renaming of the class, start with an ! character.

  It defaults to `"LinksPageSizesField_Current"` which is found in **~/DynamicData/Content/DESDynamicDataControls.css**.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Layout Pattern Properties

These properties are found in the "Layout Pattern" category of Properties window. These properties determine the positioning of the controls of this Field and HTML around them.

- **Layout** (string) – Determines the HTML and positions of controls that lay out this Field. It uses tokens to represent where the command controls appear.

Token	Description
{SIZES}	The location where the controls for Page sizes are shown. When using a DropDownPageSizesField, this is replaced by the DropDownList. When using a LinkPageSizesField, this is replaced by a list of hyperlinks separated by **LinksSeparator**.
{IMAGE}	The position for an optional image. It is used when the **ImageUrl** property is assigned.

  Omit any token of a command that you do not want to show.

  Do not use the left or right bracket characters as literal text. They are reserved for tokens. Instead, use `{` and `}`.

  The default is `"Records shown: {SIZES}"`.

- **LayoutLookupID** (string) – Gets the value for **Layout** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **Layout** 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 "".

- **ImageUrl** (string) – The URL to an image file. It is used when the "{IMAGE}" token is declared in the **Layout** property, allowing you to easily add an image to layout. The image replaces the token.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  It defaults to "" .

- **ImageAltText** (string) – When the browser cannot load the Image specified by **ImageUrl**, this text is displayed. It forms the `alt=` attribute. It is always written out. It defaults to "".

- **ImageAltTextLookupID** (string) – Gets the value for **ImageAltText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **ImageAltText** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties on DropDownSorterField and LinksSorterField

For an overview of these Field classes, see "Using the DropDownSorterField and LinksSorterField"

The `PeterBlum.DES.DynamicData.DropDownSorterField` and `PeterBlum.DES.DynamicData.LinkSorterField` inherit from `PeterBlum.DES.DynamicData.BaseControlPanelField`. For properties from its base class, see "Properties common to all Field Classes".

Click on any of these topics to jump to them:

- Properties common to all Field Classes
- SortKeys Properties
- Appearance Properties
- Layout Pattern Properties

SortKeys Properties

These properties are found in the "SortKeys" category of Properties window.

- **AutoGenerateSortKeys** (Boolean) – Determines if the sort keys list is generated using metadata, including the SortableColumnAttribute and certain data field types.

  When `true`, it uses metadata, including the SortableColumnAttribute and the property type (integer, floating, date, time, and strings with a short maximum length).

  When `false`, it uses the **SortKeys** string, which is a pipe delimited list, or if adding programmatically, calls to `AddSortKey()`.

  It defaults to `true`.

- **SortKeys** (string) – A pipe (|) delimited list of the sort keys offered, in the desired order. It should only include the actual data field names, not the Display Name. Display names will be displayed to the user.

  Any spaces around the numbers and commands will be trimmed before usage.

  When unassigned, make sure **AutoGenerateSortKeys** is `true` or you call the `AddSortKey()` method to define the list of sort keys.

  It defaults to `""`.

- **SortableMaxLength** (integer) – If the Data Field is a string type and is not assigned a SortableColumnAttribute, this is the maximum length of the data field supporting that string that allows sorting.

  DataFields that do not specify their maximum length are ignored by this property and must use the SortableColumnAttribute if you want them to sort.

  When 0, it uses a global value from **SortableColumnAttribute.SortableMaxLength** which defaults to 100. That default can be changed by assigning a new value in the `Application_Start()` method.

  It defaults to 0, and therefore uses the global default.

- **ShowNonSortableKeys** (enum PeterBlum.DES.TrueFalseIgnore) – Determines what happens when a requested data field name is found to be non-sortable either because it has SortableColumnAttribute.Sortable=`false` or its not a sortable type.

  The enumerated type TrueFalseIgnore has these values:

  - `True` – The data field is included as a sortable field.
  - `False` – The data field is omitted and an exception is thrown to point out the error. This is the default.
  - `Ignore` – The data field is omitted.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Appearance Properties

These properties are found in the "Appearance" category of Properties window.

- **NoSortKeyLabel** (string) – Text shown to indicate that no sort key is used. Not used when **NoSortKeyPosition**=`None`.

  The value is placed first in the list unless **NoSortKeyPosition**=`Last`.

  It defaults to `"None"`.

- **NoSortKeyLabelLookupID** (string) – Gets the value for **NoSortKeyLabel** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **NoSortKeyLabel** 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 "".

- **NoSortKeyPosition** (enum PeterBlum.DES.DynamicData.NoSortKeyPosition) – Determines if and where the NoSortKeyLabel appears.

  The enumerated type NoSortKeyPosition has these values:

  - `None` – Do not use the NoSortKeyLabel.
  - `First` – Show it before the sort keys. This is the default.
  - `Last` – Show it after the sort keys.

For DropDownPageSizesField

- **DropDownListCssClass** (string) – The style sheet class applied to DropDownList.

  To use automatic browser renaming of the class, start with an ! character.

  It defaults to `""`.

For LinksPageSizesField

- **LinkSeparator** (string) – The text between each hyperlink. It defaults to `" "`.
- **LinkCssClass** (string) – The style sheet class applied to each hyperlink.

  To use automatic browser renaming of the class, start with an ! character.

  It defaults to `"LinksSorterField_Link"` which is found in **~/DynamicData/Content/DESDynamicDataControls.css**.

- **CurrentSizeCssClass** (string) – The style sheet class applied to the label for the current sort key.

  To use automatic browser renaming of the class, start with an ! character.

  It defaults to `"LinksSorterField_Current"` which is found in **~/DynamicData/Content/DESDynamicDataControls.css**.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Layout Pattern Properties

These properties are found in the "Layout Pattern" category of Properties window. These properties determine the positioning of the controls of this Field and HTML around them.

- **Layout** (string) – Determines the HTML and positions of controls that lay out this Field. It uses tokens to represent where the command controls appear.

Token	Description
{SORTKEYS}	The location where the controls for sort keys are shown. When using a DropDownSorterField, this is replaced by the DropDownList. When using a LinkSorterField, this is replaced by a list of hyperlinks separated by **LinksSeparator**.
{IMAGE}	The position for an optional image. It is used when the **ImageUrl** property is assigned.

  Omit any token of a command that you do not want to show.

  Do not use the left or right bracket characters as literal text. They are reserved for tokens. Instead, use `{` and `}`.

  The default is `"Sort by: {SORTKEYS}"`.

- **LayoutLookupID** (string) – Gets the value for **Layout** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **Layout** 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 "".

- **ImageUrl** (string) – The URL to an image file. It is used when the "{IMAGE}" token is declared in the **Layout** property, allowing you to easily add an image to layout. The image replaces the token.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  It defaults to "" .

- **ImageAltText** (string) – When the browser cannot load the Image specified by **ImageUrl**, this text is displayed. It forms the `alt=` attribute. It is always written out. It defaults to "".

- **ImageAltTextLookupID** (string) – Gets the value for **ImageAltText** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. If no match is found OR this is blank, **ImageAltText** will be used.

  The String Lookup System lets you define a common set of terms so the programmer doesn't uniquely define them each time. It also provides localization based on the current culture.

  To use it, define a LookupID and associated textual value in your data source (resource, database, etc). Assign the same LookupID to this property.

  It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

AddSortKey method

Adds an entry to a list of Sort Keys. The list is internal, not the SortKeys property. Elements of this list are added before any items from the SortKeys property.

There are two forms of this method.

`AddSortKey(datafieldname, sortdirection)` adds a sort key given a data field name. It replaces the data field name with it's DisplayName. It uses the ShowNonSortableKeys property to determine what happens when a data field that isn't known to be sortable is added.

`AddSortKey(displayname, sortexpression, sortdirection)` allows adding anything, given a display name and the Sort Expression.

Both

[C#]

```
void AddSortKey(string pDataFieldName,
  PeterBlum.DES.DynamicData.ExpandedSortDirection pSortDirection)
```

```
void AddSortKey(string pDisplayName, string pSortExpression,
  PeterBlum.DES.DynamicData.ExpandedSortDirection pSortDirection)
```

[VB]

```
Sub AddSortKey(ByVal pDataFieldName As String, _
  ByVal pSortDirection As PeterBlum.DES.DynamicData.ExpandedSortDirection)
```

```
Sub AddSortKey(ByVal pDisplayName As String, ByVal pSortExpression As String, _
  ByVal pSortDirection As PeterBlum.DES.DynamicData.ExpandedSortDirection)
```

Parameters

pDataFieldName

The name of a data field to offer as sortable. This should be the internal name, not the display name.

pDisplayName

The text to show the user that identifies the sort key rule.

pSortExpression

The sort expression to pass to the DataBound field through its "Sort" command. Often the expression is the data field name (not the displayname) but it may have more complex SQL syntax.

pSortDirection

The enumerated type ExpandedSortDirection has these values:

- `Ascending` – Always sort in ascending direction.
- `Descending` – Always sort in descending direction.
- `Reverse` – Switch the direction from the current.
- `Current` – Continue with the current direction.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties on CommandButtonField and NewCommandButtonField

For an overview of these Field classes, see "Using the DropDownSorterField and LinksSorterField"

The `PeterBlum.DES.DynamicData.CommandButtonField` and `PeterBlum.DES.DynamicData.NewCommandButtonField` inherit from `PeterBlum.DES.DynamicData.BaseControlPanelField`. For properties from its base class, see "Properties common to all Field Classes".

Click on any of these topics to jump to them:

- Properties common to all Field Classes
- Appearance Properties
- Behavior Properties

Appearance Properties

These properties are found in the "Appearance" category of Properties window.

- **Text** (string) – The text shown in the button.

  For NewCommandButtonField, it defaults to `"New"`. For CommandButtonField, it defaults to "".

- **TextLookupID** (string) – Gets the value for **Text** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of Misc. 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 "".

- **ButtonType** (enum System.Web.UI.WebControls.ButtonType) – The type of button.
  - `Button` - A push button
  - `Link` - A Link button. It supports a blend of text and images by using the **LinkButtonLayout** property.
  - `Image` - An Image button. Requires the **ImageUrl** property to be assigned.

  The actual contents of the button are determined by the **Text** and **ImageUrl** properties.

  It defaults to `ButtonType.Link`.

- **ImageUrl** (string) – The optional image for the button. The URL to an image that is shown when the **ButtonType** is `Image` or `Link`.

  If this is a path to a file on your local hard disk ("/image.gif" instead of http://www.somewhereelse.com/image.gif") you can also supply a second image file to use when the Button is pressed. Simply create a file in the same path with the word "Pressed" added before the file extension.

  For example, if this is "/Images/Button.gif", also have "/Images/ButtonPressed.gif" Similarly, a mouse over is created by having an image file with the same path and "MouseOver" added before the extension.

  Supports the use of the tilde (~) as the first character to be replaced by the virtual path to the web application.

  The token "{APPEARANCE}" will be replaced by the actual location of the **Appearance** folder, which defaults to "~/DES/Appearance/", but can be overridden in the **web.config** file.

  Supports a pipe delimited list of URLs for normal|pressed|mouseover when the **MultipleImages** property is `false`. *Note: This feature requires a license covering the Peter's Interactive Pages module.*

  It defaults to `""`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **ButtonCssClass** (string) – The style sheet class applied to each button.

  To use automatic browser renaming of the class, start with an ! character.

  It defaults to `"CommandButtonField_Buttons"` which is found in **~/DynamicData/Content/DESDynamicDataControls.css**.

- **LinkButtonLayout** (enum PeterBlum.DES.DynamicData.LinkButtonLayout) – When using **ButtonType**=`Link`, it can show either or both text and image. This determines the actual layout. When set to `Auto`, only one of the two are shown. Otherwise both are shown.

  The enumerated type LinkButtonLayout has these values:

  - `Auto` - Only one is shown. If the ImageUrl is defined, an image is shown. Otherwise the text is shown.
  - `ImageOutside` - Both are shown. On the First and Previous button, the image is to the left of the text. On the Next and Last button, the image is to the right of the text. This option uses **LinkButtonLayoutSeparator** between the two elements.
  - `ImageInside` - Both are shown. On the First and Previous button, the image is to the right of the text. On the Next and Last button, the image is to the left of the text.This option uses **LinkButtonLayoutSeparator** between the two elements.

  It defaults to `LinkButtonLayout.Auto`.

- **LinkButtonLayoutSeparator** (string) – HTML placed in between an image and textual LinkButton when **ButtonType**=`Link` and **LinkButtonLayout** indicates `ImageInside` or `ImageOutside`.

  It defaults to `" "`.

- **MultipleImages** (Boolean) – When true, you have images for pressed and mouseover effects. The **ImageUrl** points to the normal graphic. Pressed and MouseOver are files in the same folder with the "Pressed" and "MouseOver" inserted in name, just before the file extension.

  The code will detect the presence of these files. If neither are found, it is not used. If one is found, it is used and the other is not. So it is safe to set this to `true` all of the time, except for the extra overhead of time used.

  The **PeterBlum.DES.Globals.Page.EnableButtonImageEffects** property can override this behavior.

  Note: This feature requires a license covering the Peter's Interactive Pages module.

  When false, it does not look for multiple images. However, the **ImageUrl** can override this by having a pipe delimited list of URLs in this format: `Normal|Pressed|MouseOver`.

  When `true`, it looks for multiple images.

  It defaults to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Behavior Properties

These properties are found in the "Behavior" category of Properties window.

- **CommandName** (string) – The Command Name sent to the DataBound control, such as "New", "Insert", or "Cancel". This property must be assigned. When using NewCommandButtonField, it is already assigned to "New" and does not need to be reassigned.
- **CommandArgument** (string) – The argument sent with the command name to the DataBound control. Its value depends on upon the command name.

  It defaults to "".
- **ConfirmMessage** (string) – Displays a confirm messagebox with this text after the button is clicked. If the user clicks No, the page will not be submitted.

  Note: This feature requires a license covering the Peter's Interactive Pages module.

  It defaults to "".
- **ConfirmMessageLookupID** (string) – Gets the value for **ConfirmMessage** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of ConfirmMessages. 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 "".
- **DisableOnSubmit** (Boolean) – Determines if the button is disabled after the page is submitted. When `true`, it is disabled.

  Note: This feature requires a license covering the Peter's Interactive Pages module.

  It defaults to `false`.
- **MayMoveOnClick** (Boolean) – If the button requires an extra click to submit the page, its because it jumped as the user clicks on it. Set this to `true` to avoid that extra click.

  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 the ValidationSummary, the button may jump. This happens before the button's onclick event, preventing that onclick event to run because the mouse button is no longer on the button.

  When `true`, the feature is enabled.

  It defaults to `false`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties on DynamicButtonsField

For an overview of this Field class, see "Using the DynamicButtonsField"

The `PeterBlum.DES.DynamicData.DynamicButtonsField` inherits from `PeterBlum.DES.DynamicData.BaseControlPanelField`. For properties from its base class, see "Properties common to all Field Classes".

It utilizes the actual DynamicButtons control. For its properties, see "Properties for the DynamicButtons Control".

Click on any of these topics to jump to them:

- Properties common to all Field Classes
- Dynamic Buttons Properties

Dynamic Buttons Properties

These properties are found in the "Dynamic Buttons" category of Properties window.

- **DynamicButtons** (PeterBlum.DES.DynamicData.DynamicButtons) – A reference to the DynamicButtons control that will output this field's contents.

  The ASP.NET Markup looks like this:

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
    <desDD:DynamicButtonsField>
      <DynamicButtons ButtonType="Image"
        NewButtonImageUrl="~/Images/NewButton.gif"
        EditButtonImageUrl="~/Images/EditButton.gif"
        DeleteButtonImageUrl="~/Images/DeleteButton.gif" />
    </desDD:DynamicButtonsField>
  </Fields>
</desDD:DynamicControlPanel>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties on TemplateControlPanelField

For an overview of this Field class, see "Using the TemplateControlPanelField"

The `PeterBlum.DES.DynamicData.TemplateControlPanelField` inherits from `PeterBlum.DES.DynamicData.BaseControlPanelField`. For properties from its base class, see "Properties common to all Field Classes".

Click on any of these topics to jump to them:

- Properties common to all Field Classes
- Appearance Properties
- Behavior Properties
- Event Handlers

Appearance Properties

These properties are found in the "Appearance" category of Properties window.

- **Template** (ITemplate) – Define the HTML and controls for your own user interface.

  Use the TemplateControlPanelField object to display navigation controls that enable users to page through data that is displayed by a DataBound control.

  You can also use the TemplateControlPanelField object to display information about the underlying data source, such as total number of records and the current page number.

  The TemplateControlPanelField class has no built-in layout. Therefore, you must explicitly create the layout within this property. You can format the content by using cascading style sheets (CSS) classes or inline style elements.

  You can reference the DynamicControlPanel control that contains the TemplateControlPanelField object by using the Container property. This is useful when you want to create binding expressions to show how many records were retrieved, the total number of pages, and similar information. Here are the values available when using Container:

Container property	Description
StartRowIndex	Gets the index of the first record that is displayed on a page of data .
TotalRowCount	Gets the total number of records that are retrieved by the underlying data source object that is referenced by the associated data-bound control.
MaximumRows	Gets the maximum number of records that are displayed for each page of data.
PageSize	Gets the number of records that are displayed for each page of data.

Example using the Container property

This example uses the event handlers shown on the previous pages.

```
<desDD:DynamicControlPanel ID="CP1" runat="server" PagedControlID="ListView1" >
  <Fields>
  <desDD:TemplateControlPanelField     IncludesPagingControls="true">
    <Template>
      Page
      <asp:Label runat="server" ID="CurrentPageLabel"
        Text="<%# Container.TotalRowCount > 0 ? (Container.StartRowIndex /
                Container.PageSize) + 1 : 0 %>" />
      of
      <asp:Label runat="server" ID="TotalPagesLabel"
        Text="<%# Math.Ceiling ((double)Container.TotalRowCount  /
             Container.PageSize) %>" />
    </Template>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

```
    </desDD:TemplateControlPanelField>
  </Fields>
  </desDD:DynamicControlPanel>
```

If your controls post back with a CommandName and CommandArgument (such as available on buttons), a default event handler will respond to these commands:

- "First", "Prev", "Next", "Last" – All attempt to change the current page index.
- "PageSize" – Change the maximum rows shown. It requires the new page size as an integer in the command argument.
- "PageIndex" – Selects a page by its number, starting at 0. The command argument must have that page index number.

If these are not how you want to handle things, use the **PagerCommand** event.

The **ContainerTag**, **HTMLBefore**, and **HTMLAfter** properties are all supported. They will enclose your template if setup.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Behavior Properties

These properties are found in the “Behavior” category of Properties window.

- **IncludesPagingControls** (Boolean) – Indicates if your template contains paging oriented controls, like next and previous commands. When so, the DynamicControlPanel's **OnePageBehavior** and **EditModeBehavior** act on it when a Paging value is selected.

  This TemplateControlPanelField may be hidden, replaced with HTML, or disabled based on the DynamicControlPanel's **OnePageBehavior** and **EditModeBehavior**.

  It defaults to `false`.

- **AutoDisable** (Boolean) – Determines if it searches through the Template's controls for buttons, textboxes, and other data entry controls and disables them if needed.

  It disables buttons when the DynamicControlPanel's **OnePageBehavior** and **EditModeBehavior** require it; also when a Button has a **CommandName** of "First", "Next", "Prev", or "Last" and the current Page is at the first or last page.

  It defaults to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Event Handlers

See also "Event Handlers" from the BaseControlPanelField class.

- **PagingStateChanged** – Called events are handled (the HandleEvents method is called) and changes were made to the current page or page size. Use this event update the controls contained in the Template, such as disabling different buttons.

  The *ControlPanelFieldEventArgs* parameter provides access to the container for the controls of this template, the PagingState, the TemplateControlPanelField, and DynamicControlPanel. See "ControlPanelFieldEventArgs Class".

  Use the `FindControl()` method to get a control by its ID to modify. Also use the **Container** property directly to modify its Controls collection.

- **PagerCommand** – Handles any command sent by a control contained in the Template. If it does not handle the command or is not setup, default handling is provided for these commands: "First", "Prev", "Next", "Last", "PageIndex", and "PageSize".

  The method definition is below.

  If you handle the command to change the starting index or maximum rows,assign the new value to **Args.NewStartRowIndex** or **Args.NewMaximumRows**. This will cause tell the caller to run `DynamicControlPanel.SetPageProperties()`.

  If you want to handle a call to DynamicControlPanel.SetPageProperties or handle any other command, you should not set NewStartRowIndex and NewMaximumRows. Instead set **Args.Handled** to `true`.

Method Declaration

[C#]

```
void MethodName(object pSender,
  PeterBlum.DES.DynamicData.DataPagerCommandEventArgs pArgs)
```

[VB]

```
Sub MethodName(ByVal pSender As Object,
  ByVal pArgs As PeterBlum.DES.DynamicData.DataPagerCommandEventArgs)
```

Parameters

pSender

The Field class. This value is also available and correctly typecast in pArgs.Field.

pArgs

Properties and methods to assist you. They are described below.

DataPagerCommandEventArgs Class

This class is passed as the argument parameter of the two event handlers described here. Here are its parameters:

- **CommandName** (string) – Gets the name of the command.
- **CommandArgument** (string) – Gets the argument associated with the command. Its value depends on the command.
- **TotalRowCount** (integer) – The total number of rows shown. It may be less than the Maximum Rows on the last page shown.
- **NewStartRowIndex** (integer) – Sets the index of the first record to display on a page of data.
- **NewMaximumRows** (integer) – Sets the maximum number of records to display on each page of data.
- **Handled** (Boolean) – Set to `true` if you have handled the event and need no further processing. If you use **NewStartRowIndex** or **NewMaximumRow** and want the caller to pass them to `DynamicControlPanel.SetPageProperties()`, leave this set to `false`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **PagingState** (PeterBlum.DES.DynamicData.BaseControlPanelField.PagingState) – Gets a reference to a class that describes the current paging state of the DataBound control. It has these properties and methods to assist you:
  - **StartRowIndex** (integer) – The index of the first record shown.
  - **MaximumRows** (integer) – The maximum number of rows that can be shown.
  - **TotalRowCount** (integer) – The total number of rows shown. It may be less than the Maximum Rows on the last page shown.
  - `GetTotalPages()` – Method. Returns an integer. The Total number of pages available.
  - `GetLastPageIndex()` – Method. Returns an integer. The page index to the last page. If there are 0 pages, it returns -1.
  - `GetPrevPageIndex()` – Method. Returns an integer. The page index to the previous page from the current one. The first page is 0.
  - `GetNextPageIndex()` – Method. Returns an integer. The page index to the next page from the current one.
  - `GetCurrentPageIndex()` – Method. Returns an integer. The page index currently shown. If there are no records, it returns -1.
  - `GetLastRowIndex()` – The index to the last record itself. (StartRowIndex + TotalRowCount – 1)
- **Item** (System.Web.UI.Control) – Gets a reference to the control that hosts the controls created by the Field class. Call FindControl("ID") to return a control by its ID.
- **TemplateControlPanelField** (PeterBlum.DES.DynamicData.TemplateControlPanelField) – Gets a reference to the TemplateControlPanelField class.
- **DynamicControlPanel** (PeterBlum.DES.DynamicData.DynamicControlPanel) – Gets a reference to the DynamicControlPanel control that owns the Field.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Constructors of the various Field Classes

[C#]

```
NextPreviousPagerField()

NextPreviousPagerField(System.Web.UI.WebControls.ButtonType pButtonType)

NextPreviousPagerField(System.Web.UI.WebControls.ButtonType pButtonType,
   string pLayout)

PreviousPagerField()

PreviousPagerField(System.Web.UI.WebControls.ButtonType pButtonType)

PreviousPagerField(System.Web.UI.WebControls.ButtonType pButtonType,
   string pLayout)

NextPagerField()

NextPagerField(System.Web.UI.WebControls.ButtonType pButtonType)

NextPagerField(System.Web.UI.WebControls.ButtonType pButtonType,
   string pLayout)

NumericPagerField()

NumericPagerField(string pLayout)

NumericPagerField(System.Web.UI.WebControls.ButtonType pNextPrevButtonType,
   string pLayout)

NumericPagerField(int pMaxPageNumbersShown)

NumericPagerField(System.Web.UI.WebControls.ButtonType pNextPrevButtonType,
   string pLayout, int pMaxPageNumbersShown)

IncrementExtendedNumericPagerField()

IncrementExtendedNumericPagerField(string pLayout)

IncrementExtendedNumericPagerField(
   System.Web.UI.WebControls.ButtonType pNextPrevButtonType,
   string pLayout)

IncrementExtendedNumericPagerField(int pMaxPageNumbersShown,
   int pExtMaxPageNumbersShown, int pIncrement, bool pRoundToNearest)

IncrementExtendedNumericPagerField(
   System.Web.UI.WebControls.ButtonType pNextPrevButtonType,
   string pLayout, int pMaxPageNumbersShown, int pExtMaxPageNumbersShown,
   int pIncrement, bool pRoundToNearest)


SpreadExtendedNumericPagerField()

SpreadExtendedNumericPagerField(string pLayout)

SpreadExtendedNumericPagerField(
   System.Web.UI.WebControls.ButtonType pNextPrevButtonType,
   string pLayout)

SpreadExtendedNumericPagerField(int pMaxPageNumbersShown,
   int pExtMaxPageNumbersShown, int pMinimumBetweenPages)

SpreadExtendedNumericPagerField(
   System.Web.UI.WebControls.ButtonType pNextPrevButtonType,
   string pLayout, int pMaxPageNumbersShown, int pExtMaxPageNumbersShown,
   int pMinimumBetweenPages)
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

```
JumpByExtendedNumericPagerField()

JumpByExtendedNumericPagerField(string pLayout)

JumpByExtendedNumericPagerField(
  System.Web.UI.WebControls.ButtonType pNextPrevButtonType,
  string pLayout)

JumpByExtendedNumericPagerField(int pJumpBy)

JumpByExtendedNumericPagerField(
  System.Web.UI.WebControls.ButtonType pNextPrevButtonType,
  string pLayout, int pJumpBy)

InfoPagerField()

InfoPagerField(string pLayout)

InfoPagerField(string pLayout, bool pEditablePageIndex)

DropDownPageSizesField()

DropDownPageSizesField(string pSizes)

DropDownPageSizesField(string pSizes, string pLayout)

LinksPageSizesField()

LinksPageSizesField(string pSizes)

LinksPageSizesField(string pSizes, string pLayout)

DropDownSorterField()

DropDownSorterField(string pSortKeys)

DropDownSorterField(string pSortKeys, string pLayout)

LinksSorterField()

LinksSorterField(string pSortKeys)

LinksSorterField(string pSortKeys, string pLayout)

NewCommandButtonField()

NewCommandButtonField(System.Web.UI.WebControls.ButtonType pButtonType)

NewCommandButtonField(System.Web.UI.WebControls.ButtonType pButtonType,
  string pText, string pImageUrl)

CommandButtonField()

CommandButtonField(System.Web.UI.WebControls.ButtonType pButtonType,
  string pCommandName)

CommandButtonField(System.Web.UI.WebControls.ButtonType pButtonType,
  string pCommandName,
  string pText, string pImageUrl)

DynamicButtonsField()

DynamicButtonsField(System.Web.UI.WebControls.ButtonType pButtonType)

TemplateControlPanelField()

TemplateControlPanelField(ITemplate pTemplate)

TemplateControlPanelField(ITemplate pTemplate, bool pIncludesPagingFeatures)
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

[VB]

NextPreviousPagerField

```
New()
New(ByVal pButtonType As System.Web.UI.WebControls.ButtonType)
New(ByVal pButtonType As System.Web.UI.WebControls.ButtonType,
  ByVal pLayout As String)
```

PreviousPagerField

```
New()
New(ByVal pButtonType As System.Web.UI.WebControls.ButtonType)
New(ByVal pButtonType As System.Web.UI.WebControls.ButtonType,
  ByVal pLayout As String)
```

NextPagerField

```
New()
New(ByVal pButtonType As System.Web.UI.WebControls.ButtonType)
New(ByVal pButtonType As System.Web.UI.WebControls.ButtonType,
  ByVal pLayout As String)
```

NumericPagerField

```
New()
New(ByVal pLayout As String)
New(ByVal pNextPrevButtonType As System.Web.UI.WebControls.ButtonType,
  ByVal pLayout As String)
New(ByVal pMaxPageNumbersShown As Integer)
New(ByVal pNextPrevButtonType As System.Web.UI.WebControls.ButtonType,
  ByVal pLayout As String, ByVal pMaxPageNumbersShown As Integer)
```

IncrementExtendedNumericPagerField

```
New()
New(ByVal pLayout As String)
New(ByVal pNextPrevButtonType As System.Web.UI.WebControls.ButtonType,
  ByVal pLayout As String)
New(ByVal pMaxPageNumbersShown As Integer,
  ByVal pExtMaxPageNumbersShown As Integer,
  ByVal pIncrement As Integer,
  ByVal pRoundToNearest As Boolean)
New(ByVal pNextPrevButtonType As System.Web.UI.WebControls.ButtonType,
  ByVal pLayout As String, ByVal pMaxPageNumbersShown As Integer,
  ByVal pExtMaxPageNumbersShown As Integer,
  ByVal pIncrement As Integer,
  ByVal pRoundToNearest As Boolean)
```

SpreadExtendedNumericPagerField

```
New()
New(ByVal pLayout As String)
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

```
New(ByVal pNextPrevButtonType As System.Web.UI.WebControls.ButtonType,
  ByVal pLayout As String)
```

```
New(ByVal pMaxPageNumbersShown As Integer,
  ByVal pExtMaxPageNumbersShown As Integer, ByVal pMinimumBetweenPages As Integer)
```

```
New(ByVal pNextPrevButtonType As System.Web.UI.WebControls.ButtonType,
  ByVal pLayout As String, ByVal pMaxPageNumbersShown As Integer,
  ByVal pExtMaxPageNumbersShown As Integer,
  ByVal pMinimumBetweenPages As Integer)
```

JumpByExtendedNumericPagerField

```
New()
```

```
New(ByVal pLayout As String)
```

```
New(ByVal pNextPrevButtonType As System.Web.UI.WebControls.ButtonType,
  ByVal pLayout As String)
```

```
New(ByVal pJumpBy As Integer)
```

```
New(ByVal pNextPrevButtonType As System.Web.UI.WebControls.ButtonType,
  ByVal pLayout As String, ByVal pJumpBy As Integer)
```

InfoPagerField

```
New()
```

```
New(ByVal pLayout As String)
```

```
New(ByVal pLayout As String, ByVal pEditablePageIndex As Boolean)
```

DropDownPageSizesField

```
New()
```

```
New(ByVal pSizes As String)
```

```
New(ByVal pSizes As String, ByVal pLayout As String)
```

LinksPageSizesField

```
New()
```

```
New(ByVal pSizes As String)
```

```
New(ByVal pSizes As String, ByVal pLayout As String)
```

DropDownSorterField

```
New()
```

```
New(ByVal pSortKeys As String)
```

```
New(ByVal pSortKeys As String, ByVal pLayout As String)
```

LinksSorterField

```
New()
```

```
New(ByVal pSortKeys As String)
```

```
New(ByVal pSortKeys As String, ByVal pLayout As String)
```

NewCommandButtonField

```
New()
```

```
New(ByVal pButtonType As System.Web.UI.WebControls.ButtonType)
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

```
New(ByVal pButtonType As System.Web.UI.WebControls.ButtonType,
  ByVal pText As String, ByVal pImageUrl As String)
```

CommandButtonField

```
New()
```

```
New(ByVal pButtonType As System.Web.UI.WebControls.ButtonType,
  ByVal pCommandName As String)
```

```
New(ByVal pButtonType As System.Web.UI.WebControls.ButtonType,
  ByVal pCommandName As String,
  ByVal pText As String, ByVal pImageUrl As String)
```

DynamicButtonsField

```
New()
```

```
New(ByVal pButtonType As System.Web.UI.WebControls.ButtonType)
```

TemplateControlPanelField

```
New()
```

```
New(ByVal pTemplate As ITemplate)
```

```
New(ByVal pTemplate As ITemplate, ByVal pIncludesPagingFeatures As Boolean)
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Field Templates: The user interface for columns

A Field Template defines the user interface for each data type of the columns in your database. Your web application contains many Field Templates, with at least one for each data type used by your application. There are Field Templates for integers, decimals, dates, times, strings, and many more.

Click on any of these topics to jump to them:

- Overview
- Maintaining separation of concerns
- Field Templates in the DES/FieldTemplates folder RadControls Ajax Controls Toolkit
- Creating and modifying Field Templates
- Setting Properties on the Field Templates
- Field Template classes and properties
- Using Style Sheets within Field Templates
- Customization based on Categories
- Implementing the ICustomizeFieldTemplate Interface
- FieldTemplateFactory: Selecting the correct Field Template file

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Overview

- Field Templates are User Controls that inherit from `PeterBlum.DES.DynamicData.DESFieldTemplateUserControl` or one of its subclasses. See "Field Template classes and properties".
- They often expose public properties that the web form developer can use to customize their behavior. See "Setting Properties on the Field Templates".
- Field Templates are defined for their usage: readonly (non-editable), edit mode, and insert mode. Their file names tell you exactly what they are for by including the data type and if in edit or insert mode, "_Edit" or "_Insert". For example, "Date.ascx", "Date_Edit.ascx" and "Date_Insert.ascx". If your user interface for edit and insert modes is identical, you can omit declaring an insert mode template. Dynamic Data will automatically fall back to using the edit mode template.
- The Field Templates shipped with DES Dynamic Data are located in the **[web application]/DynamicData/DES/FieldTemplates** folder. For a list of them, see "Field Templates in the DES/FieldTemplates folder".
- If you are using Telerik RadControls, you can also install Field Templates that utilize various RadControls. For details, see "Field Templates in the RadControls/FieldTemplates folder".
- If you are using Ajax Control Toolkit, you can also install Field Templates that utilize some of its controls. For details, see "Field Templates in the Ajax Controls Toolkit folder".
- Use the **FieldTemplates.config** file to determine which Field Template to use when two of the same name exist in different folders. It also lets you create an alias that maps the requested Field Template name to another. See "FieldTemplateFactory: Selecting the correct Field Template file".
- If DES Dynamic Data lacks a Field Template or needs a different user interface for an existing one, you can create one by modifying an existing one or creating your own. See "Creating and modifying Field Templates".
- You can define one or more for a specific data type. For example, the data type DateTime may have user interfaces to edit just the date, just the time, and both. You may have one that uses a DateTextBox and another that uses a Calendar control.
- To be selected, the file name of the User Control must match one of these:
  - The actual data type of the column. For example, a DateTime property will have the file name **DateTime.ascx**.
  - The type specified by a DataTypeAttribute in the metadata. For example, [DataType(DataType.Url)] will use the file name **Url.ascx**.
  - The name specified in the UIHintAttribute in the metadata. For example, [UIHint("Calendar")] will use the file name **Calendar.ascx**.
  - The name specified in the **UIHint** property on the DynamicControl or DynamicField. For example, `<desDD:DynamicField UIHint="Calendar" />` will use the file name **Calendar.ascx**.
- When applying style sheet classes to the controls and HTML within your Field Template, define them in the **[web app]/DynamicData/Contents/FieldTemplates.css** file and the **AutoLoadFieldTemplateStyleSheetFile** property to `true` in `Page_Init()`. See "Using Style Sheets within Field Templates".
- If you want to modify the controls of a Field Template at runtime, setup the ICustomizeFieldTemplate Interface. See "Implementing the ICustomizeFieldTemplate Interface".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Maintaining separation of concerns

As you know, ASP.NET Dynamic Data separates web form development into three areas: the business logic, Field Templates, and web forms. Here is how these affect the Field Templates.

- Business logic – Metadata can be used to select Field Templates through the DataTypeAttribute and UIHintAttribute. The DataTypeAttributes will pick a default Field Template. The UIHintAttribute will pick a non-default Field Template. (If neither are assigned to the column, the column's datatype picks the default Field Template.) Generally the business logic should not explicitly pick a Field Template. That should be left to the web form developer. So limit using the UIHintAttribute.
- Web forms – The web form developer can override the default Field Template and its properties by using properties on the DynamicControl and DynamicField. They can also implement the ICustomizeFieldTemplate interface to programmatically modify the properties.
- Field Templates – Actual Field Template development includes modifing existing templates and creating new ones. Since a change to an existing Field Template will effect web forms that are already using that template, care must be taken to avoid breaking those web forms. See "Creating and modifying Field Templates".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Field Templates in the DES/FieldTemplates folder

Here are the Field Templates shipped with DES Dynamic Data in the **DES/FieldTemplates** folder. Click on the Template Name to view properties you can set with a UIHintAttribute, DynamicField, or DynamicControl.

The source code for the base classes used by these Field Templates is found in **[DES Product Folder]/DynamicData/FieldTemplates Base Classes**.

Template name	DataType	DataControl type	Column Metadata	Notes
Anniversary.ascx	DateTime	Label	DES.AnniversaryDataTypeAttribute	
Anniversary_Edit.ascx	DateTime	AnniversaryTextBox	DES.AnniversaryDataTypeAttribute	
Boolean.ascx	Boolean, Integer	Image	Boolean: Automatic Integer: UIHint("Boolean")	Selects between two images
BooleanCheckBox.ascx	Boolean, Integer	Checkbox	UIHint("BooleanCheckBox")	Checkbox is disabled
BooleanLabel.ascx	Boolean, Integer	Label	UIHint(ReadOnlyUIHint= "BooleanLabel”)	Selects between two text values
Boolean_Edit.ascx	Boolean, Integer	Checkbox	Boolean: Automatic Integer: UIHint("Boolean")	Integer uses 0 for false. All others are true. When saving, it only saves 0 or 1.
Calendar_Edit.ascx	DateTime	Calendar (DES)	UIHint("Calendar")	
Children.ascx	One to many	Hyperlink	Automatic	Links to a Page Template showing the contents of a table
Currency.ascx	Double, Decimal	Label	DES.CurrencyDataTypeAttribute	
Currency_Edit.ascx	Double, Decimal	CurrencyTextBox	DES.CurrencyDataTypeAttribute	
Date.ascx	DateTime	Label	DES.DateDataTypeAttribute	
Date_Edit.ascx	DateTime	DateTextBox	DES.DateDataTypeAttribute	
DateTime.ascx	DateTime	Label	Automatic	
DateTime_Edit.ascx	DateTime	TimeOfDayTextBox	Automatic	Also has DateTextBox
DbImage.ascx	Binary	Image	DES.DbImageDataTypeAttribute	Optionally pops up a full size image.
DbImage_Edit.ascx	Binary	FileInput and Image	DES.DbImageDataTypeAttribute	Optionally pops up a full size image.
Decimal.ascx	Double, Decimal	Label	Automatic. Optional: DES.DecimalDataTypeAttribute	
Decimal_Edit.ascx	Double, Decimal	DecimalTextBox	Automatic. Optional: DES.DecimalDataTypeAttribute	

CONTINUED ON THE NEXT PAGE

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Click on the Template Name to view properties you can set with a UIHintAttribute, DynamicField, or DynamicControl.

Template name	DataType	DataControl type	Column Metadata	Notes
Duration.ascx	DateTime, TimeSpan Integer, Double	Label	DES.DurationDataTypeAttribute	
Duration_Edit.ascx	DateTime, TimeSpan, Integer, Double	DurationTextBox	DES.DurationDataTypeAttribute	
Enumerated.ascx	Integer, String	Label	Integer: DES.EnumeratedAttribute String: DES.EnumeratedStringAttribute	
Enumerated_Edit.ascx	Integer, String	DropDownList	Integer: DES.EnumeratedAttribute String: DES.EnumeratedStringAttribute	
ForeignKey.ascx	One to one	HyperLink	Automatic	Links to a Page Template showing the details of a record.
ForeignKey_Edit.ascx	One to one	DropDownList	Automatic	
Integer.ascx	Integer	Label	Automatic. Optional: DES.IntegerDataTypeAttribute	
Integer_Edit.ascx	Integer	IntegerTextBox	Automatic. Optional: DES.IntegerDataTypeAttribute	
ManyToMany.ascx	Many to many	ListView of DynamicHyperlinks	Automatic	Only installed when using Entity Framework
ManyToMany_Edit.ascx	Many to many	CheckBoxList	Automatic	Only installed when using Entity Framework
Measurement.ascx	Integer, Double, Decimal	Label	DES.IntegerMeasurementDataTypeAttribute DES.DecimalMeasurementDataTypeAttribute	Use MeasurementAttribute
Measurement_Edit.ascx	Integer, Double, Decimal	IntegerTextBox or DecimalTextBox	DES.IntegerMeasurementDataTypeAttribute DES.DecimalMeasurementDataTypeAttribute	Use MeasurementAttribute. Selects the textbox by the column type
MonthYear.ascx	DateTime	Label	DES.MonthYearDataTypeAttribute	
MonthYear_Edit.ascx	DateTime	MonthYearTextBox	DES.MonthYearDataTypeAttribute	
MultilineText_Edit.ascx	String	TextBox	DataType(DataType.MultilineText)	Textbox is TextMode=Multiline. Offers a TextCounter control. When the mode is read-only, it uses Text.ascx.

CONTINUED ON THE NEXT PAGE

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Click on the Template Name to view properties you can set with a UIHintAttribute, DynamicField, or DynamicControl.

Template name	DataType	DataControl type	Column Metadata	Notes
Password.ascx	String	Label	DataType(DataType.Password)	Shows "*****" instead of the password
Password_Edit.ascx	String	FilteredTextBox	DataType(DataType.Password)	Textbox is TextMode=Password. Use CharacterSetAttribute to apply a filter.
Percent.ascx	Integer, Double, Decimal	Label	DES.PercentDataTypeAttribute	
Percent_Edit.ascx	Double, Decimal	PercentTextBox	DES.PercentDataTypeAttribute	
Text.ascx	String	Label	Automatic	
Text_Edit.ascx	String	FilteredTextBox	Automatic	Use CharacterSetAttribute to apply a filter.
Time.ascx	DateTime	Label	DES.TimeOfDayDataTypeAttribute	
Time_Edit.ascx	DateTime	TimeOfDayTextBox	DES.TimeOfDayDataTypeAttribute	
Url.ascx	String	Label	DataType(DataType.Url)	
Url_Edit.ascx	String	TextBox	DataType(DataType.Url)	
UrlAsImage.ascx	String	Image and Label	DES.UrlAsImageDataTypeAttribute	Optionally pops up a full size image.
UrlAsImage_Edit.ascx	String	Image and TextBox	DES.UrlAsImageDataTypeAttribute	Optionally pops up a full size image.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Field Templates in the RadControls/FieldTemplates folder

Here are the Field Templates shipped with DES Dynamic Data in the **RadControls/FieldTemplates** folder. They are subclassed from classes defined in the source code file **RadControls_DynamicData.cs/vb**, which is usually found in the **App_Code/DES** folder.

Template name	DataType	DataControl type	Column Metadata	Notes
Currency_Edit.ascx	Double, Decimal	RadNumericTextBox	DES.CurrencyDataTypeAttribute	
Date_Edit.ascx	DateTime	RadDatePicker	DES.DateDataTypeAttribute	
DateTime_Edit.ascx	DateTime	RadDateTimePicker	Automatic	
Decimal_Edit.ascx	Double, Decimal	RadNumericTextBox	DES.DecimalDataTypeAttribute	
Enumerated_Edit.ascx	Integer, String	RadCombobox	Integer: DES.EnumeratedAttribute String: DES.EnumeratedStringAttribute	
ForeignKey_Edit.ascx		RadCombox	Automatic	
Html_Edit.ascx	String	RadEditor	DataType(DataType.Html)	
Integer_Edit.ascx	Integer	RadNumericTextBox	Automatic. Optional: DES.IntegerDataTypeAttribute	
Measurement_Edit.ascx	Integer, Double, Decimal	RadNumericTextBox	DES.IntegerMeasurementDataTypeAttribute DES.DecimalMeasurementDataTypeAttribute	
MultilineText_Edit.ascx	String	RadTextBox	DataType(DataType.MultilineText)	TextMode=Multline. If using HTML, use Html_Edit.ascx
Percent_Edit.ascx	Integer, Double, Decimal	RadNumericTextBox	DES.PercentDataTypeAttribute	
Text_Edit.ascx	String	RadTextBox	Automatic	
Time_Edit.ascx	DateTime	RadTimePicker	DES.TimeOfDayDataTypeAttribute	

All Field Template classes offer properties for you to further customize them through the UIHintAttribute, DynamicField, and DynamicControl. See "Setting Properties on the Field Templates" and the properties shown in the **RadControls_DynamicData.cs/vb** file, which is usually found in the **App_Code/DES** folder.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Field Templates in the Ajax Controls Toolkit folder

Here are the Field Templates shipped with DES Dynamic Data in the **Ajax Controls Toolkit** folder. They are subclassed from classes defined in the source code file **AJAXControlToolKit_DynamicData.cs/vb**, which is usually found in the **App_Code/DES** folder. They are optionally installed.

Template name	DataType	DataControl type	Column Metadata	Notes
DateAjaxCalendar_Edit.ascx	DateTime	TextBox + CalendarExtender	DES.DateDataTypeAttribute	
IntegerSlider_Edit.ascx	Integer	TextBox + SliderExtender	Automatic	

All Field Template classes offer properties for you to further customize them through the UIHintAttribute, DynamicField, and DynamicControl. See "Setting Properties on the Field Templates" and the properties shown in the **AJAXControlToolKit_DynamicData.cs/vb** file, which is usually found in the **App_Code/DES** folder.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Creating and modifying Field Templates

Field Templates display and sometimes edit the data of one column in a table. They provide a way to standardize the user interface. DES supplies numerous Field Templates, found in the **[web application]/DynamicData/FieldTemplates** folder. Sometimes you will want to modify their user interface. Before you directly edit them, remember that the change will apply to all columns for the datatype supported by that Field Template. It may be better to create another Field Template, with a new name and use the UIHintAttribute, DynamicField or DynamicControl to select it on a case-by-case basis.

There are several ways to introduce new Field Templates.

- Copy an existing template, using a new file name.
- Create a file that uses one of DES's FieldTemplateUserControl classes.

Copy an Existing Template

Note: The Field Templates supplied by DES do not use a code behind file to simplify the process of copying existing templates.

1. *Optional*. Create a new folder where you keep your custom Field Templates. Generally your folder should go in the **[web application]/DynamicData** folder or a child folder of **DynamicData/FieldTemplates**.

   When you do this, add a `<path>` tag to the `<pathsearch>` section **FieldTemplates.config** file. See "FieldTemplateFactory: Selecting the correct Field Template file".

   In this example, the user has created **[web application]/DynamicData/CustomFieldTemplates**.

```
<pathsearch>
  <path url="Field Templates"   />
  <path url="CustomFieldTemplates"  />
  <path url="Field Templates/RadControls"  />
  <path url="" />
</pathsearch>
```

2. Copy and rename the desired template. It should still have an ".ascx" file extension. When using step 1, above, copy it into the folder for your custom templates.
3. Edit the controls in the new Field Template.
4. Use the UIHintAttribute on the EntityTable Metadata Class or **UIHint** property on the DynamicField or DynamicControl to select the new Field Template. See "UIHintAttribute: Customizing the Field Template used".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using one of DES's FieldTemplateUserControl classes

Open any of DES's Field Templates and you will see very little program code. That's because most of the code has been built into the class identified by the user control's **Inherits** property. Here is the Field Template for Integer_Edit.ascx:

```
<%@ Control Language="C#" Inherits="PeterBlum.DES.DynamicData.IntegerEditFTUC" %>
<%@ Register tagprefix="desDD" assembly="PeterBlum.DES.DynamicData"
      namespace="PeterBlum.DES.DynamicData" %>
<script runat="server">
   protected void Page_Init(object sender, EventArgs e)
   {
      SetUpEditableDataControl(IntegerTextBox1);
      SetUpColumnValidatorManager(ColumnValidatorManager1);
   }
</script>
<des:IntegerTextBox ID="IntegerTextBox1" runat="server" Columns="10" />
<desDD:ColumnValidatorManager ID="ColumnValidatorManager1" runat="server" />
```

In most cases, you can create Field Templates that are just as simple. All it takes is inheriting from one of the following classes (all in the `PeterBlum.DES.DynamicData` namespace):

DataType	Mode	Class	DataControl type
Anniversary	ReadOnly	AnniversaryStaticFTUC	Label
Anniversary	Edit/Insert	AnniversaryEditFTUC	AnniversaryTextBox
Binary (image)	ReadOnly	DBImageStaticFTUC	Image
Binary (image)	Edit/Insert	DBImageEditFTUC	FileUpload and Image
Boolean	ReadOnly	BooleanImageStaticFTUC	Image
Boolean	ReadOnly	BooleanCheckBoxStaticFTUC	CheckBox
Boolean	ReadOnly	BooleanLabelStaticFTUC	Label
Boolean	Edit/Insert	BooleanCheckBoxEditFTUC	CheckBox
Currency	ReadOnly	CurrencyStaticFTUC	Label
Currency	Edit/Insert	CurrencyEditFTUC	CurrencyTextBox
Date	ReadOnly	DateStaticFTUC	Label
Date	Edit/Insert	DateEditFTUC	DateTextBox
Date	Edit/Insert	CalendarEditFTUC	DES Calendar
DateTime	ReadOnly	DateTimeStaticFTUC	Label
DateTime	Edit/Insert	DateTimeEditFTUC	DateTextBox and TimeOfDayTextBox
Decimal	ReadOnly	DecimalStaticFTUC	Label
Decimal	Edit/Insert	DecimalEditFTUC	DecimalTextBox
Duration	ReadOnly	DurationStaticFTUC	Label
Duration	Edit/Insert	DurationEditFTUC	DurationTextBox
Enumerated	ReadOnly	EnumeratedStaticFTUC	Label
Enumerated	Edit/Insert	EnumeratedEditFTUC	DropDownList or ListBox
ForeignKey	ReadOnly	ForeignKeyStaticFTUC	Label and Hyperlink
ForeignKey	Edit/Insert	ForeignKeyEditFTUC	DropDownList or ListBox
Integer	ReadOnly	IntegerStaticFTUC	Label
Integer	Edit/Insert	IntegerEditFTUC	IntegerTextBox

CONTINUED ON THE NEXT PAGE

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

DataType	Mode	Class	DataControl type
ManyToMany	ReadOnly	EFManyToManyStaticFTUC	ListView or Repeater
ManyToMany	Edit (insert not used)	EFManyToManyEditFTUC	CheckBoxList or multiple selection ListBox
Measurement	ReadOnly	MeasurementStaticFTUC	Labels
Measurement	Edit/Insert	MeasurementEditFTUC	DecimalTextBox and IntegerTextBox
MonthYear	ReadOnly	MonthYearStaticFTUC	Label
MonthYear	Edit/Insert	MonthYearEditFTUC	MonthYearTextBox
MultilineText	Edit/Insert	MultilineTextEditFTUC	DES TextBox
Password	Edit/Insert	TextEditFTUC	FilteredTextBox
Percent	ReadOnly	PercentStaticFTUC	Label
Percent	Edit/Insert	PercentEditFTUC	PercentTextBox
Text	Edit/Insert	TextEditFTUC	FilteredTextBox
Text (Url)	ReadOnly	UrlStaticFTUC	Label
Text (Url)	Edit/Insert	UrlEditFTUC	FilteredTextBox
Text (Url to Image)	ReadOnly	UrlAsImageStaticFTUC	Image
Text (Url to Image)	Edit/Insert	UrlAsImageEditFTUC	Image
TimeOfDay	ReadOnly	TimeOfDayStaticFTUC	Label
TimeOfDay	Edit/Insert	TimeOfDayEditFTUC	TimeOfDayTextBox

For any DataType/Mode combination that is not listed above, inherit from `PeterBlum.DES.DynamicData.DESFieldTemplateUserControl`. This is a base class for all of the above classes and has a much richer featureset than its ancestor, `System.Web.DynamicData.FieldTemplateUserControl`. When you do, you will override two methods, `AssignData()` and `ExtractValues()`. See the last two steps of the instructions that follow.

The source code for the Field Template classes is found in **[DES Product Folder]/DynamicData/FieldTemplates Base Classes**. It is only in C#.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Steps for inheriting from a DES FieldTemplateUserControl class

1. *Optional*. Create a new folder where you keep your custom Field Templates. Generally your folder should go in the [web application]/DynamicData folder or a child folder.

   When you do this, add a `<path>` tag to the `<pathsearch>` section **FieldTemplates.config** file. See "FieldTemplateFactory: Selecting the correct Field Template file".

   In this example, the user has created **[web application]/DynamicData/CustomFieldTemplates**.

```
<pathsearch>
  <path url="Field Templates" />
  <path url="CustomFieldTemplates" />
  <path url="Field Templates/RadControls" />
  <path url="" />
</pathsearch>
```

2. Create a User Control with a name that matches the datatype or UIHint label you want. It should go in a Field Templates folder, either **[web application]/DynamicData/FieldTemplates** or the one created in step 1.

3. Modify the `<%@ Control %>` tag to declare the **Inherits** property with the full type name of the FieldTemplateUserControl class you want to use.

   For example:

```
<%@ Control Language="C#" Inherits="PeterBlum.DES.DynamicData.IntegerEditFTUC" %>
```

4. If this template will support validators, add this declaration to the top of the file:

```
<%@ Register tagprefix="desDD" assembly="PeterBlum.DES.DynamicData"
    namespace="PeterBlum.DES.DynamicData" %>
```

5. Add the `Page_Init()` method. At this time, do not add any code to it.

   [C#]

```
protected void Page_Init(object sender, EventArgs e)
{
}
```

   [VB]

```
Protected Sub Page_Init(ByVal sender As Object, ByVal e As EventArgs)

End Sub
```

6. Add the controls that display the data, such as textboxes, labels and lists. Remember that the mode of the Field Template should determine if the controls support editing or not.

7. If this Field Template supports validators, add DES's ColumnValidatorManager control where you want to see inline validation errors.

   - Some of the ColumnValidatorManager properties can be set using the Customizer control.
   - You do not need to any Validator controls that are supported by ValidationAttributes. However, if you want to fully control a validator's properties, add it too. It will be attached to the ColumnValidatorManager in a later step.

```
<desDD:ColumnValidatorManager ID="ColumnValidatorManager1" runat="server" />
```

   For more details, see "How Field Templates utilize the ValidationAttributes".

8. In `Page_Init()`, call either `SetupStaticDataControl()` or `SetupEditableDataControl()`, passing the control that displays/edits data. If the Field Template supports several, call the method for each of them.

```
SetupStaticDataControl(Label1)

SetupEditableDataControl(TextBox1)
```

CONTINUED ON THE NEXT PAGE

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

9. When using the ColumnValidatorManager control, in `Page_Init()`, call `SetUpColumnValidatorManager()` passing the ColumnValidatorManager control.

```
SetUpColumnValidatorManager(ColumnValidatorManager1)
```

10. When using a validator control, in `Page_Init()`, call `SetUpDESValidator()` passing the validator control. Always call it after the call to `SetUpColumnValidatorManager()` to connect it to the ColumnValidatorManager.

```
SetUpDESValidator(RequiredListValidator1)
```

11. When inheriting from `PeterBlum.DES.DynamicData.DESFieldTemplateUserControl`, create the `AssignData()` method. It overrides an existing method and is passed one parameter, FieldValue. Write the code that applies FieldValue to the controls of your Field Template.

    The property **DataControl** contains a reference to the first control passed with `SetupStaticDataControl()` or `SetupEditableDataControl()`.

Example: ReadOnly mode

This will convert the FieldValue to a string and assign it to a label.

[C#]

```
protected override void AssignData(object pFieldValue)
{
   ((TextBox)DataControl).Text = pFieldValue.ToString();
}
```

[VB]

```
Protected Overrides Sub AssignData(ByVal pFieldValue As Object)
   CType(DataControl, TextBox).Text = pFieldValue.ToString()
End Sub
```

Example: Edit or Insert mode

This will assign a boolean FieldValue to a checkbox's Checked property.

[C#]

```
protected override void AssignData(object pFieldValue)
{
   if (pFieldValue is bool)
      ((CheckBox)DataControl).Checked = (bool) pFieldValue;
   else
      throw new Exception("Data is not compatible with the Field Template");
}
```

[VB]

```
Protected Overrides Sub AssignData(ByVal pFieldValue As Object)
   If TypeOf pFieldValue Is Boolean Then
      CType(DataControl, CheckBox).Checked= CType(pFieldValue, Boolean)
   Else
      Throw New Exception("Data is not compatible with the Field Template")
   End If
End Sub
```

CONTINUED ON THE NEXT PAGE

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

12. When inheriting from `PeterBlum.DES.DynamicData.DESFieldTemplateUserControl` and the template supports editing, create the `ExtractValues()` method. It overrides an existing method and is passed one parameter, Dictionary. Write the code that applies the FieldValue from the controls to `Dictionary[Column.Name]`.

    The property **DataControl** contains a reference to the first control passed with `SetupStaticDataControl()` or `SetupEditableDataControl()`.

    Example

    This will get the state of the CheckBox and assign it to the dictionary.

    [C#]

    ```
    protected override void ExtractValues(IOrderedDictionary dictionary)
    {
       dictionary[Column.Name] = CheckBox1.Checked;
    }
    ```

    [VB]

    ```
    Protected Overrides Sub ExtractValues(ByVal dictionary As IOrderedDictionary)
       dictionary[Column.Name] = CheckBox1.Checked
    End Sub
    ```

13. Your Field Template can make recommendations for styles and attributes applied to a container control, such as alignment when its placed in a `<td>` tag. It collects these styles and attributes within its **ContainerTagAttributes** property. The `PeterBlum.DES.DynamicData.ContainerTagAttributes` class has these properties: **Width**, **Wrap** (when `false`, it prevents word wrap), **HorizontalAlign**, and **VerticalAlign**.

    Override the `SetDefaultContainerTagAttributes()` method in your Field Template class to assign any of these properties.

    [C#]

    ```
    protected override void SetDefaultContainerTagAttributes(
       PeterBlum.DES.DynamicData.ContainerTagAttributes pContainerTagAttributes)
    {
       pContainerTagAttributes.Wrap = false;
    }
    ```

    [VB]

    ```
    Protected Overrides Sub SetDefaultContainerTagAttributes( _
       ByVal pContainerTagAttributes As _
       PeterBlum.DES.DynamicData.ContainerTagAttributes)
       pContainerTagAttributes.Wrap = False
    End Sub
    ```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Setting Properties on the Field Templates

Each Field Template is derived from a class, and that class offers public properties to customize its behavior. For example, the base class used by all DES Field Templates, `PeterBlum.DES.DynamicData.DESFieldTemplateUserControl`, has these public properties: **DataControlWidth**, **DataControlHeight**, **UseDescriptionAsToolTip**, **UseDependencyAttribute**, and **ValueWhenBlank**. See "Field Template classes and properties".

Since DynamicControls are used to load Field Templates, you cannot directly set these properties. Instead, you can use any of these tools to set them:

- DynamicControl – Used within the templates of your DataBound controls and Pattern Templates to load a single Field Template for a specific data field. See "Supply alternative values for properties on the Field Template".
- DataFieldInPattern object – Used with the DynamicPattern, DynamicListView, and DynamicFormView controls to setup a Field Template for a specific data field. See "Supply alternative values for properties on the Field Template".
- DynamicField – Used in the Fields collection of the DetailsView and Columns collection of the GridView to specify a column using ASP.NET Dynamic Data. See "Supply alternative values for properties on the Field Template".
- UIHintAttribute – Define properties within your business logic using metadata on the column in the EntityTable Metadata Class. See "Setting properties ".
- DataTypeAttributes – Some DataTypeAttributes offer properties that are associated with the Field Template they load.
- IFieldGeneratorGuidance Interface – When using automatic scaffolding, you can programmatically create DataFieldInPattern objects with the Field Template properties they pass along. See "Customizing Automatic Scaffolding".
- ICustomizeFieldTemplates Interface - Lets you modify properties on a Field Template at runtime. See "Implementing the ICustomizeFieldTemplate Interface".

In addition, the Customizer Control has numerous properties that are respected by the Field Templates.

Choosing the right place to set properties on the Field Template

Most of the tools have the same set of properties where you assign a list of Field Template property names and values. They differ in when to use them.

- When working with a DynamicControl, DynamicField object, or DataFieldInPattern object that has a specific data field assigned, use its properties.
- When you want the business logic to set the properties, use either the UIHintAttribute or, if the DataTypeAttribute has a corresponding property, the DataTypeAttribute. However, remember Field Templates are part of the user interface layer. If you are trying to maintain a separation between business logic and user interface, consider other choices.
- You can define Field Template files and assign their properties in the `Page_Init()` method. These settings will be overrriden by properties on any of the previously mentioned techniques. Create several versions of your Field Template files and use the **UIHint** property on UIHintAttribute, DataFieldInPattern, DynamicControl, or DynamicField to select the desired file.
- When you need to programmatically set a property within you page's code, use the ICustomizeFieldTemplates Interface.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Field Template classes and properties

DES provides many base classes for developing Field Templates. See "Using one of DES's FieldTemplateUserControl classes". Many classes provide properties that let you customize their behavior. They can be set by using the UIHintAttribute, DynamicField and DynamicControl. See "Setting Properties on the Field Templates". This section describes those properties. (It does not cover the properties you might use as you develop subclasses of `PeterBlum.DES.DynamicData.DESFieldTemplateUserControl`.)

Many of these properties modify the **DataControl**, a property on DESFieldTemplateUserControl that refers to the control showing and optionally editing the data, such as the TextBox.

Click on any of these topics to jump to them:

- Properties Common to All Field Templates
- Text.ascx Field Template
- Text_Edit.ascx and MultilineText_Edit.ascx Field Templates
- Integer.ascx, Decimal.ascx, Currency.ascx, Percent.ascx Field Templates
- Integer_Edit.ascx, Decimal_Edit.ascx, Currency_Edit.ascx, Percent_Edit.ascx Field Templates
- Measurement.ascx and Measurement_Edit.ascx Field Templates
- All Date, DateTime, MonthYear, and Anniversary Field Templates
- Time.ascx and Time_Edit.ascx Field Templates
- Duration.ascx and Duration_Edit.ascx Field Templates
- Calendar_Edit.ascx Field Template
- Enumerated.ascx Field Template — Enumerated_Edit.ascx Field Template
- Boolean.ascx Field Template — Boolean_Edit.ascx Field Template
- BooleanLabel.ascx Field Template — BooleanCheckBox.ascx Field Template
- DbImage.ascx and DbImage_Edit.ascx Field Templates
- UrlAsImage.ascx and UrlAsImage_Edit.ascx Field Templates
- Url.ascx Field Template — Url_Edit.ascx Field Template
- EmailAddress.ascx Field Template — EmailAddress_Edit.ascx Field Template
- ForeignKey.ascx Field Template — ForeignKey_Edit.ascx Field Template

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties Common to All Field Templates

Field Template Base class	PeterBlum.DES.DynamicData.DESFieldTemplateUserControls
Field Template files	All subclass directly or through one of the classes described in the remaining topics

See "Setting Properties on the Field Templates".

- **DataControl** (System.Web.UI.Control) – Gets a reference to the control registered on this Field Template that displays or edits data. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `DataControl-propertyname='value'`. For example:

  ```
  <des:DynamicField DataField="LastName"
   EditPropertyList="DataControl-Width='100px'" />
  ```

  Subclassed Field Templates often declare another property to the same control, with the actual control type, such as "TextBox", "Label", or "Image". You can use either to set properties.

- **DataControlWidth** (System.Web.UI.WebControls.Unit) – Sets the **Width** property on the DataControl. When set to **Unit.Empty**, is is not used. It defaults to **Unit.Empty**.

  Field Templates that use a TextBox also provide the ability to set the **Columns** property, which affects the width. **DataControlWidth** overrides that setting.

- **DataControlHeight** (System.Web.UI.WebControls.Unit) – Sets the **Height** property on the DataControl. When set to **Unit.Empty**, is is not used. It defaults to **Unit.Empty**.

  Field Templates that use a TextBox also provide the ability to set the **Rows** property, which affects the height. **DataControlHeight** overrides that setting.

- **UseDescriptionAsToolTip** (enum PeterBlum.DES.TrueFalseDefault) – The **MetaColumn.Description** value can be assigned to the ToolTip of the DataControl when this is True or Default and Customizer.UseDescriptionAsToolTip is true. The Customizer actually has three settings, which depend on the mode: ReadOnly, Edit, or Insert. They are **UseDescriptionAsToolTipReadOnly**, **UseDescriptionAsToolTipEdit**, and **UseDescriptionAsToolTipInsert**.

  When `TrueFalseDefault.True`, use the description.

  When `TrueFalseDefault.False`, do not use the description.

  When `TrueFalseDefault.Default`, use the properties from the Customizer. If the Customizer is not on the page, Edit and Insert modes will setup the ToolTip. ReadOnly mode will not.

  It defaults to `TrueFalseDefault.Default`.

  Tooltips can become Enhanced Tooltips (using PopupViews) by setting up the HintManager.EnableToolTipsUsePopupViews property on the PageManager control.

- **UseDependencyAttribute** (enum PeterBlum.DES.TrueFalseDefault) – Determines if any DependencyAttribute on this column is used to disable or hide this column and to turn off its validators.

  When `TrueFalseDefault.True`, the DependencyAttribute is used.

  When `TrueFalseDefault.False`, it is not. The data entry control is always active along with its validators.

  When `TrueFalseDefault.Default`, it gets the value from the **Customizer.UseDependencyAttributes** property. If the Customizer is not on the page, the DependencyAttribute is used.

  It defaults to `TrueFalseDefault.Default`.

- **ValueWhenBlank** (string) – When the database supplies no value or a blank string, this value can be shown instead.

  In readonly mode, subclasses can display this value. It can contain HTML.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

In edit or insert mode, it sets the **ValueWhenBlank** property on DES TextBoxes, which displays this text in the textbox when the value is blank. It automatically supports any Field Template whose DataControl property is one of DES's (any subclassed from PeterBlum.DES.BaseTextBox).

It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Text.ascx Field Template

Field Template Base class	PeterBlum.DES.DynamicData.TextStaticFTUC
Field Template files	Text.ascx
DataTypes supported	String Also displays the string from almost any datatype, when that type does not have a matching Field Template.
DataControl types	Label, Panel, LiteralControl, PlaceHolder, or System.Web.UI.HtmlControls.HtmlGenericControl. Others if you subclass. When using PlaceHolder, it adds a LiteralControl with the content to your PlaceHolder.

Writes the string into the innerHTML of the control supplied. Usually that control is a Label, but it can be a Panel or. (See also "Supporting other controls to show the text".) It applies HtmlEncoding unless the HtmlEncode property overrides this behavior.

Properties

See "Setting Properties on the Field Templates".

- **HtmlEncode** (PeterBlum.DES.TrueFalseDefault) – Determines if HtmlEncoding is applied. By default, HtmlEncoding is applied.

  The enumerated type `PeterBlum.DES.TrueFalseDefault` has these values:

  - `True` – Always HtmlEncode. Let the user see HTML tags as text.
  - `False` – Never HtmlEncode. Let the browser process HTML tags.
  - `Default` – Use the value of the HtmlEncode property found on the DynamicField or DynamicControl object that requested this template. If not explicitly declared on those controls, they always HtmlEncode. This is the default.

- **ShowUrlsAsHyperlinks** (PeterBlum.DES.TrueFalseDefault) – Determines if URLs within the text are converted to hyperlinks.

  The enumerated type `PeterBlum.DES.TrueFalseDefault` has these values:

  - `True` – Convert to hyperlinks.
  - `False` – Do not convert.
  - `Default` – Looks a the Customizer control for a value in the **Text_ShowUrlsAsHyperlinks** property. If there is no Customizer or **Text_ShowUrlsAsHyperlinks** itself is `TrueFalseDefault.Default`, use the business rules on the column's metadata. It will convert to hyperlinks when the DataTypeAttribute is set to Multiline or the length is very long (or not defined at all). This is the default.

- **ShowEmailsAsHyperlinks** (PeterBlum.DES.TrueFalseDefault) – Determines if email addresses within the text are converted to hyperlinks.

  The enumerated type `PeterBlum.DES.TrueFalseDefault` has these values:

  - `True` – Convert to hyperlinks.
  - `False` – Do not convert.
  - `Default` – Looks a the Customizer control for a value in the **Text_ShowEmailsAsHyperlinks** property. If there is no Customizer or **Text_ShowEmailsAsHyperlinks** itself is `TrueFalseDefault.Default`, use the business rules on the column's metadata. It will convert to hyperlinks when the DataTypeAttribute is set to Multiline or the length is very long (or not defined at all). This is the default.

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Supporting other controls to show the text

`PeterBlum.DES.DynamicData.TextStaticFTUC` supports only Label, Panel, and HtmlGenericControl to show the text in its innerHtml. Your Field Template is a subclass of TextStaticFTUC, so it can override this limitation, supplying additional control types.

Override these two methods:

- `SupportedDataControls(control)` – Passed a control. Return `true` if its type is supported.

```
protected override bool SupportedDataControls(Control pDataControl)
```

```
Protected Overrides Function SupportedDataControls( _
 ByVal pDataControl As Control) As Boolean
```

- `AssignText(string)` – Passed the string (already HtmlEncoded if needed). Assign it to your control.

```
protected override void AssignText(string pFieldValueString)
```

```
Protected Overrides Sub AssignText(ByVal pFieldValueString As String)
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Text_Edit.ascx and MultilineText_Edit.ascx Field Templates

Field Template Base classes	PeterBlum.DES.DynamicData.TextEditFTUC PeterBlum.DES.DynamicData.MultilineTextEditFTUC
Field Template files	Text_Edit.ascx, MultilineText_Edit.ascx
DataTypes supported	String
DataControl types	System.Web.UI.WebControls.TextBox and its subclasses

Properties

See "Setting Properties on the Field Templates".

- **TextBox** (System.Web.UI.WebControls.TextBox) – Gets a reference to the TextBox control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `TextBox-propertyname='value'`. For example:

```
<des:DynamicField DataField="LastName"
 EditPropertyList="TextBox-CssClass='myTextBoxCss'" />
```

- **Columns** (integer) – Sets the **Columns** property on the TextBox. Not used when 0. It defaults to 0.

  When 0, the Customizer control's **Text_Columns** property is used. It defaults to 80.

  This is overridden by setting the **DataControlWidth** property.

- **Rows** (integer) – Sets the **Rows** property on the TextBox. Not used when 0. It defaults to 0.

  When 0, the Customizer control's **Text_Rows** property is used. It defaults to 5.

  This is overridden by setting the **DataControlHeight** property.

- **AutoConvertCase** (boolean) – DES TextBoxes offer the **ConvertCase** property. When **AutoConvertCase** is `true`, if there is a CharacterSetAttribute where only upper or lower case is allowed, it automatically sets the **TextBox.ConvertCase** property.

- **TextCounter** (PeterBlum.DES.TextCounter) – Gets a reference to the optional TextCounter control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `TextCounter-propertyname='value'`. For example:

```
<des:DynamicField DataField="LastName"
 EditPropertyList="TextCounter-AboveMaximumCssClass='DES_TCCAboveMax'" />
```

  For details on the TextCounter, see the **Interactive Pages User's Guide**.

  The Customizer control determines if the TextCounter is shown with its **UseTextCounter** property. By default, it is set to show when the column has a maximum length. MultilineText_Edit.ascx is the only Field Template that includes a TextCounter control. You can add it to Text_Edit.ascx and others that subclass from `PeterBlum.DES.DynamicData.BaseTextBoxFTUC`, `BaseTextAreaFTUC`, `TextAreaFTUC`, and `MultilineTextEditFTUC` classes. If you do so, call this method in `Page_Init()` to register your TextCounter.

```
SetUpTextCounter(TextCounter1)
```

  If you want to ignore the **Customizer.UseTextCounter** property and enable or disable it in your Field Template, override the `UseTextCounter()` method to return `true` (to enable) or `false` (to disable).

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Integer.ascx, Decimal.ascx, Currency.ascx, Percent.ascx Field Templates

Field Template Base classes	PeterBlum.DES.DynamicData.IntegerStaticFTUC PeterBlum.DES.DynamicData.DecimalStaticFTUC PeterBlum.DES.DynamicData.CurrencyStaticFTUC PeterBlum.DES.DynamicData.PercentStaticFTUC
Field Template files	Integer.ascx, Decimal.ascx, Currency.ascx, Percent.ascx
DataTypes supported	Integer, Floating point
DataControl types	Label

Properties

No properties introduced.

See "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Integer_Edit.ascx, Decimal_Edit.ascx, Currency_Edit.ascx, Percent_Edit.ascx Field Templates

Field Template Base classes	PeterBlum.DES.DynamicData.IntegerEditFTUC PeterBlum.DES.DynamicData.DecimalEditFTUC PeterBlum.DES.DynamicData.CurrencyEditFTUC PeterBlum.DES.DynamicData.PercentEditFTUC
Field Template files	Integer_Edit.ascx, Decimal_Edit.ascx, Currency_Edit.ascx, Percent_Edit.ascx
DataTypes supported	Integer, Floating point
DataControl types	IntegerTextBox, DecimalTextBox, CurrencyTextBox, PercentTextBox

Properties

See "Setting Properties on the Field Templates".

- **Columns** (integer) – Sets the **Columns** property on the TextBox. Not used when 0.

  When in use, the **SmartColumns** property is not used.

  It defaults to 0.

  This is overridden by setting the **DataControlWidth** property.

- **SmartColumns** (enum PeterBlum.DES.DynamicData.SmartNumericColumns) – Determines how to set the **Columns** property on the textbox. It uses what it thinks is the largest reasonable size for the numbers expected.

  When **MaxWholeDigits** property on the DataTypeAttribute is set, it takes into account the following:

  - The **MaxWholeDigits** property.
  - Number of possible decimal places. For Decimal, it uses the **MaxDecimalPlaces** property on DES.DecimalDataTypeAttribute. For Currency, it uses the current CultureInfo.NumberFormat.CurrencyDecimalPlaces. For decimal Percentages, it uses 2.
  - Use of thousands separators, currency symbol and percent symbol.
  - The minus characters when negative numbers are allowed.

  For example, when the DES.IntegerDataTypeAttribute has **MaxWholeDigits** set to 2 and allows negatives, it calculates 3 (2 digits + minus character).

  It also uses the DES.RangeAttribute. *Note: The RangeAttribute will only be used if it increases the Columns size determined by the MaxWholeDigits rules above.*

  The enumerated type `PeterBlum.DES.DynamicData.SmartNumericColumns` has these values:

  - `Never` - Do not modify the existing value of Columns.
  - `WhenZero` - Only modify it when its 0.
  - `Always` - Modify it.
  - `Default` - Use the **SmartColumns** property on the Customizer control. If there is no Customizer, it uses `WhenZero`. This is the default.

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Measurement.ascx and Measurement_Edit.ascx Field Templates

Field Template Base classes	PeterBlum.DES.DynamicData.MeasurementStaticFTUC PeterBlum.DES.DynamicData.MeasurementEditFTUC
Field Template files	Measurement.ascx, Measurement_Edit.ascx
DataTypes supported	Integer, floating point
DataControl types	Label (for ReadOnly Field Templates) IntegerTextBox and DecimalTextBox

Properties

See "Setting Properties on the Field Templates".

- **DataFormatString** (string) – Determines how to place the number and its name in a string. On Measurement.ascx, the string is assigned to the Label control. On Measurement_Edit.ascx, the {VALUE} token is used by the TextBox. Text before that token goes to the first Label and the after goes to the second Label that are on the Field Template.

  Uses the token {VALUE} for the number, {NAME} for Name, {PLURALNAME} for PluralName, and {ABBREVNAME} for AbbrevName.

  HTMLEncoding is applied based on the **HtmlEncode** property.

  When "", it uses the **DataFormatString** on the DES.IntegerMeasurementDataTypeAttribute or DES.DecimalMeasurementDataTypeAttribute.

  It defaults to "".

- **DataFormatStringLookupID** (string) – An alternative to **DataFormatString**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. Strings must be in the string group ValidationMisc.

- **HtmlEncode** (enum PeterBlum.DES.TrueFalseDefault) – Determines if names and DataFormatString are HtmlEncoded or not.

  When `TrueFalseDefault.Default`, it uses the value from the **HtmlEncode** property on the DES.IntegerMeasurementDataTypeAttribute or DES.DecimalMeasurementDataTypeAttribute.

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

All Date, DateTime, MonthYear, and Anniversary Field Templates

Field Template Base classes	PeterBlum.DES.DynamicData.DateStaticFTUC PeterBlum.DES.DynamicData.DateEditFTUC PeterBlum.DES.DynamicData.DateTimeStaticFTUC PeterBlum.DES.DynamicData.DateTimeEditFTUC PeterBlum.DES.DynamicData.MonthYearStaticFTUC PeterBlum.DES.DynamicData.MonthYearEditFTUC PeterBlum.DES.DynamicData.AnniversaryStaticFTUC PeterBlum.DES.DynamicData.AnniversaryEditFTUC
Field Template files	Date.ascx, Date_Edit.ascx, DateTime.ascx, DateTime_Edit.ascx, MonthYear.ascx, MonthYear_Edit.ascx, Anniversary.ascx, and Anniversary_Edit.ascx
DataTypes supported	DateTime
DataControl types	Label (all ReadOnly Field Templates) DateTextBox, TimeOfDayTextBox (for DateTime only) MonthYearTextBox AnniversaryTextBox

Properties

See "Setting Properties on the Field Templates".

- **DateFormat** (System.Nullable<PeterBlum.DES.EntryDateFormat>) – Determines the format for a date.

  If `null`, it uses one of these properties on the Customizer control: **DateTextBox_DateFormat**, **MonthYearTextBox_DateFormat**, **AnniversaryTextBox_DateFormat**, or **Label_DateFormat**.

  It defaults to `null`.

  The enumerated type `PeterBlum.DES.EntryDateFormat` has these values:

  - `Short` – Use the ShortDatePattern. Months are shown as digits or abbreviated month names depending on AllowMonthNames.
  - `Abbreviated` – Use the LongDatePattern. Months are always abbreviated. Weekdays are removed.
  - `Long` – Use the LongDatePattern. Months are always full. Weekdays are removed.

- **AllowMonthNames** (System.Nullable<PeterBlum.DES.AllowMonthNames>) – Customizes the entry and display of the month component in the date string when **DateFormat**=`Short`.

  If `null`, it uses one of these properties on the Customizer control: **DateTextBox_AllowMonthNames**, **MonthYearTextBox_AllowMonthNames**, **AnniversaryTextBox_AllowMonthNames**, or **Label_AllowMonthNames**.

  It defaults to `null`.

  The enumerated type `PeterBlum.DES.AllowMonthNames` has these values:

  - `No` – The user can only enter digits.
  - `Yes` – The user can enter digits or letters but the value is converted to digits.
  - `Show` – The user can enter digits or letters and the value is converted to month names.

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Time.ascx and Time_Edit.ascx Field Templates

Field Template Base classes	PeterBlum.DES.DynamicData.TimeOfDayStaticFTUC PeterBlum.DES.DynamicData.TimeOfDayEditFTUC
Field Template files	Time.ascx, Time_Edit.ascx
DataTypes supported	DateTime
DataControl types	Label TimeOfDayTextBox

Properties

None introduced.

See also “Properties Common to All Field Templates” and “Properties on the Customizer Control”.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Duration.ascx and Duration_Edit.ascx Field Templates

Field Template Base classes	PeterBlum.DES.DynamicData.DurationStaticFTUC PeterBlum.DES.DynamicData.DurationEditFTUC
Field Template files	Duration.ascx, Duration_Edit.ascx
DataTypes supported	DateTime, Integer, Double, TimeSpan
DataControl types	Label DurationTextBox

Properties

None introduced.

See also “Properties Common to All Field Templates” and “Properties on the Customizer Control”.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Enumerated.ascx Field Template

Field Template Base classes	PeterBlum.DES.DynamicData.EnumeratedStaticFTUC
Field Template files	Enumerated.ascx
DataTypes supported	Integer, string
DataControl types	Label

Properties

None introduced.

See “Properties Common to All Field Templates” and “Properties on the Customizer Control”.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Enumerated_Edit.ascx Field Template

Field Template Base classes	PeterBlum.DES.DynamicData.EnumeratedEditFTUC
Field Template files	Enumerated_Edit.ascx
DataTypes supported	Integer, string
DataControl types	DropDownList

Properties

See "Setting Properties on the Field Templates".

- **DropDownList** (System.Web.UI.WebControls.DropDownList) – Gets a reference to the DropDownList control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `DropDownList-propertyname='value'`. For example:

```
<des:DynamicField DataField="LastName"
 EditPropertyList="DropDownList-CssClass='myDDLCss'" />
```

- **RequireSelectionText** (string) – The first item of the DropDownList when the column requires data. When set to "{DEFAULT}", it looks for the Customizer. If found, it uses the value of **Customizer.ForeignKeyRequireSelectionText**. If not found, it uses "[Make a selection]". It defaults to "`{DEFAULT}`".
- **RequireSelectionTextLookupID** (string) – An alternative to **RequireSelectionText**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. Strings must be in the string group Misc.
- **NoSelectionText** (string) – The first item of the DropDownList when the column does not require data. When set to "{DEFAULT}", it looks for the Customizer. If found, it uses the value of **Customizer.ForeignKeyNoSelectionText**. If not found, it uses "[Not set]". It defaults to "`{DEFAULT}`".
- **NoSelectionTextLookupID** (string) – An alternative to **NoSelectionText**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. Strings must be in the string group Misc.
- **Sort** (boolean) – When `true`, the list is alphabetically sorted. However, the first item is always the **RequiredSelectionText** or **NoSelectionText**. It defaults to `false`.

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

ForeignKey.ascx Field Template

Field Template Base classes	PeterBlum.DES.DynamicData.ForeignKeyStaticFTUC
Field Template files	ForeignKey.ascx
DataTypes supported	One-to-one relationship (usually integer or string)
DataControl types	HyperLink and Label (both are expected on the Field Template)

Properties

See "Setting Properties on the Field Templates".

- **HyperLink** (System.Web.UI.WebControls.HyperLink) – Gets a reference to the HyperLink control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `HyperLink-propertyname='value'`. For example:

```
<des:DynamicField DataField="EmployeeID"
 EditPropertyList="HyperLink-CssClass='myCss'" />
```

- **Label** (System.Web.UI.WebControls.Label) – Gets a reference to the Label control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `Label-propertyname='value'`. For example:

```
<des:DynamicField DataField="EmployeeID"
 EditPropertyList="Label-CssClass='myCss'" />
```

- **NavigateUrl** (string) – When assigned, this URL is used by the hyperlink. When blank, the URL comes from the **ForeignKeyPath** property on the base class for all Field Templates.

  This URL will be built using the Page Template routing model through the BuildForeignKeyPath() method, which gives [routing path] + NavigateUrl + PrimaryKeyID=PrimaryKeyValue.

  For example, when NavigateUrl = "Fun?Games=Yes", PrimaryKeyID="Brand", and PrimaryKeyValue="200", this will become: MyTableName/Fun?Games=Yes&Brand=2000.

  If the user doesn't want that feature, start with "~" and give the path from the web application root folder.

  Supports the token "{TABLENAME}" which will be replaced by the Table.Name (not the DisplayName).

  Supports the token "{KEYNAME}" which is the column name containing the primary key.

  Supports the token "{KEYVALUE}" which is url encoded representation of FieldValue

  It defaults to "".

- **AllowNavigation** (enum PeterBlum.DES.TrueFalseDefault) – Determines if the UI uses the Hyperlink or Label.

  When `TrueFalseDefault.True`, use the hyperlink.

  When `TrueFalseDefault.False`, use the label.

  When `TrueFalseDefault.Default`, use the **AllowForeignKeyNavigation** property on DES's DynamicDataManager control, which when `true` uses the hyperlink.

  It defaults to `TrueFalseDefault.Default`.

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

ForeignKey_Edit.ascx Field Template

Field Template Base classes	PeterBlum.DES.DynamicData.ForeignKeyEditFTUC
Field Template files	ForeignKey_Edit.ascx
DataTypes supported	One-to-one relationship (usually integer or string)
DataControl types	DropDownList

Properties

See "Setting Properties on the Field Templates".

- **DropDownList** (System.Web.UI.WebControls.DropDownList) – Gets a reference to the DropDownList control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `DropDownList-propertyname='value'`. For example:

```
<des:DynamicField DataField="EmployeeID"
 EditPropertyList="DropDownList-CssClass='myDDLCss'" />
```

- **RequireSelectionText** (string) – The first item of the DropDownList when the column requires data. When set to "{DEFAULT}", it looks for the Customizer. If found, it uses the value of **Customizer.ForeignKeyRequireSelectionText**. If not found, it uses "[Make a selection]". It defaults to "`{DEFAULT}`".
- **RequireSelectionTextLookupID** (string) – An alternative to **RequireSelectionText**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. Strings must be in the string group Misc.
- **NoSelectionText** (string) – The first item of the DropDownList when the column does not require data. When set to "{DEFAULT}", it looks for the Customizer. If found, it uses the value of **Customizer.ForeignKeyNoSelectionText**. If not found, it uses "[Not set]". It defaults to "`{DEFAULT}`".
- **NoSelectionTextLookupID** (string) – An alternative to **NoSelectionText**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. Strings must be in the string group Misc.

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

ManyToMany.ascx Field Template

Field Template Base classes	PeterBlum.DES.DynamicData.EFManyToManyStaticFTUC
Field Template files	ManyToMany.ascx – Requires Entity Framework for .net v4
DataTypes supported	Many-to-many relationship
DataControl types	Repeater or ListView. Subclass to support others.

Properties

See "Setting Properties on the Field Templates".

These properties are declared directly in the ManyToMany.ascx Field Template because they are specific to using a ListView control.

- **ItemCountInListColumn** (integer) – When inside a list-style interface, this is the limit of the number of items (hyperlinks) per row. It defaults to 1.
- **ItemCountInDetail** (integer) – When inside a single record-style interface, this is the limit of the number of items (hyperlinks) per row. It defaults to 3.

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

ManyToMany_Edit.ascx Field Template

Field Template Base classes	PeterBlum.DES.DynamicData.ManyToManyEditFTUC
Field Template files	ManyToMany_Edit.ascx – Requires Entity Framework for .net 4
DataTypes supported	Many-to-many relationship (usually integer or string)
DataControl types	CheckBoxList or multiple selection ListBox. Subclass to support others.

Properties

See "Setting Properties on the Field Templates".

These properties are declared directly in the ManyToMany_Edit.ascx Field Template because they are specific to using a CheckBoxList control.

- **ColumnCount** (integer) – This is the limit of the number of columns output by the CheckBoxList. It defaults to 3.

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Boolean.ascx Field Template

Field Template Base classes	PeterBlum.DES.DynamicData.BooleanImageStaticFTUC
Field Template files	Boolean.ascx
DataTypes supported	Boolean, Integer
DataControl types	Image

See also BooleanLabel.ascx and BooleanCheckBox.ascx for alternative user interfaces.

Properties

See "Setting Properties on the Field Templates".

- **Image** (System.Web.UI.WebControls.Image) – Gets a reference to the Image control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `Image-propertyname='value'`. For example:

```
<des:DynamicField DataField="DriversLicense"
 ReadOnlyPropertyList="Image-Width='20px'" />
```

- **ImageUrlForTrue** (string) – The URL to the image shown when the boolean value is `true`.

  It defaults to "`{APPEARANCE}/Shared/BlueDot.gif`" ( ).

- **ImageUrlForFalse** (string) – The URL to the image shown when the boolean value is `false`.

  It defaults to "`{APPEARANCE}/Shared/spacer.gif`" which is a transparent 1x1 image.

- **AltTextForTrue** (string) – The alternative text applied to the image when the boolean value is `true`.

  It defaults to "`Yes`".

- **AltTextForFalse** (string) – The alternative text applied to the image when the boolean value is `false`.

  It defaults to "`No`".

- **AltTextForTrueLookupID** (string) – An alternative to **AltTextForTrue**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. Strings must be in the string group Misc. It defaults to "".

- **AltTextForFalseLookupID** (string) – An alternative to **AltTextForFalse**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. Strings must be in the string group Misc. It defaults to "".

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Making BooleanLabel or BooleanCheckBox the default for any readonly boolean case

In the `<aliases>` section of the **~/DynamicData/FieldTemplates.config** file, add this line:

```
<alias requested="Boolean" newname="BooleanLabel"
   namesource="DataType" modes="readonly" />
```

or

```
<alias requested="Boolean" newname="BooleanCheckBox"
   namesource="DataType" modes="readonly" />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

BooleanLabel.ascx Field Template

Field Template Base classes	PeterBlum.DES.DynamicData.BooleanLabelStaticFTUC
Field Template files	BooleanLabel.ascx
DataTypes supported	Boolean, Integer
DataControl types	Label

See also Boolean.ascx and BooleanCheckBox.ascx for alternative user interfaces.

Properties

See "Setting Properties on the Field Templates".

- **Label** (System.Web.UI.WebControls.Label) – Gets a reference to the Label control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `Label-propertyname='value'`. For example:

```
<des:DynamicField DataField="DriversLicense"
 ReadOnlyPropertyList="Label-CssClass='myCss'" />
```

- **TextForTrue** (string) – The text applied to the Label when the boolean value is `true`.

  It defaults to "`Yes`".

- **TextForFalse** (string) – The text applied to the Label when the boolean value is `false`.

  It defaults to "`No`".

- **TextForTrueLookupID** (string) – An alternative to **TextForTrue**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. Strings must be in the string group Misc. It defaults to "".

- **TextForFalseLookupID** (string) – An alternative to **TextForFalse**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. Strings must be in the string group Misc. It defaults to "".

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Example using UIHintAttribute

[C#]

```
[DES.UIHint("BooleanLabel",
  ReadOnlyPropertyList="TextForTrue='True' TextForFalse='False'")]
```

[VB]

```
<DES.UIHint("BooleanLabel", _
  ReadOnlyPropertyList:="TextForTrue='True' TextForFalse='False'")> _
```

Example using DynamicField

```
<desDD:DynamicField DataField="Discontinued" UIHint="BooleanLabel"
  ReadOnlyPropertyList="TextForTrue='True' TextForFalse='False'" />
```

Making BooleanLabel the default for any readonly boolean case

In the `<aliases>` section of the **~/DynamicData/FieldTemplates.config** file, add this line:

```
<alias requested="Boolean" newname="BooleanLabel"
  namesource="DataType" modes="readonly" />
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

BooleanCheckBox.ascx Field Template

Field Template Base classes	PeterBlum.DES.DynamicData.BooleanCheckBoxStaticFTUC
Field Template files	BooleanCheckBox.ascx
DataTypes supported	Boolean, Integer
DataControl types	CheckBox (Enabled=false)

See also Boolean.ascx and BooleanLabel.ascx for alternative user interfaces.

Properties

See "Setting Properties on the Field Templates".

- **CheckBox** (System.Web.UI.WebControls.CheckBox) – Gets a reference to the CheckBox control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `CheckBox-propertyname='value'`. For example:

```
<des:DynamicField DataField="DriversLicense"
 ReadOnlyPropertyList="CheckBox-CssClass='myCss'" />
```

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Boolean_Edit.ascx Field Template

Field Template Base classes	PeterBlum.DES.DynamicData.BooleanCheckBoxEditFTUC
Field Template files	Boolean_Edit.ascx
DataTypes supported	Boolean, Integer
DataControl types	CheckBox

Properties

See "Setting Properties on the Field Templates".

- **CheckBox** (System.Web.UI.WebControls.CheckBox) – Gets a reference to the CheckBox control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `CheckBox-propertyname='value'`. For example:

```
<des:DynamicField DataField="DriversLicense"
 EditPropertyList="CheckBox-CssClass='myCss'" />
```

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Children.ascx Field Template

Field Template Base classes	PeterBlum.DES.DynamicData.ChildrenStaticFTUC
Field Template files	Children.ascx
DataTypes supported	One-to-many relationship
DataControl types	HyperLink

Properties

See "Setting Properties on the Field Templates".

- **HyperLink** (System.Web.UI.WebControls.HyperLink) – Gets a reference to the HyperLink control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `HyperLink-propertyname='value'`. For example:

```
<des:DynamicField DataField="Employees"
 ReadOnlyPropertyList="HyperLink-CssClass='myCss'" />
```

- **NavigateUrl** (string) – The Url to assign to the HyperLink. When assigned, this is the path used. When blank, it uses the path determined the DynamicDataRoutes you've defined in `Application_Start()` to view a page template listing the records associated with this one-to-many relationship. It defaults to "".
- **AllowNavigation** (boolean) – When `true`, the HyperLink is enabled allowing the user to move to the page that lists the records associated with this one-to-many relationship. When `false`, the HyperLink is disabled (has no Url assigned). It shows the label defined by HyperLinkText on the HyperLink. It defaults to `true`.
- **HyperLinkText** (string) – The text shown by the Hyperlink control. Use the token {TABLENAME} to show the current table name. It defaults to "View {TABLENAME}".
- **HyperLinkTextLookupID** (string) – An alternative to **HyperLinkText**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. Strings must be in the string group Misc. It defaults to "".

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

DbImage.ascx and DbImage_Edit.ascx Field Templates

Field Template Base classes	PeterBlum.DES.DynamicData.DbImageStaticFTUC PeterBlum.DES.DynamicData.DbImageEditFTUC
Field Template files	DbImage.ascx, DbImage_Edit.ascx
DataTypes supported	Binary containing an image
DataControl types	Image Label (for ReadOnly Field Template) FileUpload (for Edit/Insert Field Templates)

This Field Template uses an Image control to display an image from a binary data type column. Usually you will establish maximum width and height limits to fit the image on the page. It can optionally display an "Expanded View", which shows the same image in a larger size. The Expanded View opens when the user clicks on the image.

You setup the Expanded View by assigning a Url to a specialized UserControl within the **ExpandedViewUserControlUrl** property. DES Dynamic Data includes an Expanded View that uses the Ajax Control Toolkit's Modal Popup Extender in "~/DynamicData/FieldTemplates/Ajax Control Toolkit". For more on creating other Expanded View user controls, see "Creating Expanded View UserControls".

This Field Template optionally shows text in a Label control to annotate the image. The text shown is assigned in the **LabelText** property.

When in Edit and Insert mode, `PeterBlum.DES.DynamicData.DbImageEditFTUC` supports a FileUpload control to retrieve a file. It's server side code checks that the file is actually an image from one of the expected types (determined by DES.DbImageDataTypeAttribute.SupportedFileTypes). It also calls code you can use to invoke a virus checker.

The user interface supplied by **DbImage_Edit.ascx** displays the image and FileUpload controls at various times. By attaching an Edit button, it hides the FileUpload control until the user clicks Edit. By attaching a Clear button, it hides the Image and shows the FileUpload control.

Click on any of these topics to jump to them:

- Properties on the Image
- Properties for a ToolTip
- Properties for a Label
- Properties for an Expanded View
- Properties for Editing the Image File
- Properties Common to All Field Templates
- Properties on the Customizer Control
- Creating Expanded View UserControls
- Adding Virus Detection to uploaded files

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties on the Image

See "Setting Properties on the Field Templates".

- **Image** (System.Web.UI.WebControls.Image) – Gets a reference to the Image control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `Image-propertyname='value'`. For example:

```
<des:DynamicField DataField="CustomerPhoto"
 ReadOnlyPropertyList="Image-Width='80px'" />
```

- **TimeStampColumnName** (string) – Optional name specifying a column containing a DateTime value or long integer that reflects a unique identity of the value stored in the Image's column. This allows the user to change the image and have browser's caches see that change by including an additional element in the QueryString that comes from this column.

  Field Templates based in PeterBlum.DES.DynamicData.DbImageEditFTUC will automatically support this. They update the TimeStamp column with either the current date time or DateTime converted into a long integer, depending on the column's type.

  If not supplied, no time stamp is used to monitor changes.

  The DES.DbImageDataTypeAttribute also has a **TimeStampColumnName** property. It is only used when this property is unassigned.

  It defaults to "".

- **ImageWidth** (integer) – Overrides the image's actual width with this value, in pixels. When 0, use the image's actual width. The DES.DbImageDataTypeAttribute can also specify a width using its **DisplayWidth** property. The value from **DES.DbImageDataTypeAttribute.DisplayWidth** is only used when this property is 0.

  It defaults to 0.

- **ImageHeight** (integer) – Overrides the image's actual height with this value, in pixels. When 0, use the image's actual height. The DES.DbImageDataTypeAttribute can also specify a height using its **DisplayHeight** property. The value from **DES.DbImageDataTypeAttribute.DisplayHeight** is only used when this property is 0.

  It defaults to 0.

- **MissingImageUrl** (string) – An image to show when the database does not supply an image. If not assigned when needed, the Image control is hidden.

  It defaults to `"{APPEARANCE}/Shared/Spacer.gif"` which is a transparent 1x1 gif.

- **ImageAlternativeText** (string) – The alternative text used by the image on the browser. The browser will display it as a tooltip (unless overridden by **ToolTipText**) and on the page when the actual image Url doesn't return an image. It defaults to "".

- **ImageAlternativeTextLookupID** (string) – An alternative to **ImageAlternativeText**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) Strings must be in the string group Misc. It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties for a ToolTip

See "Setting Properties on the Field Templates".

- **ToolTipText** (string) – Text shown in the tooltip. The tooltip can be an Enhanced Tooltip (a feature of the Peter's Interactive Pages module that uses a PopupView to display an HTML-driven tooltip.) You must set ToolTipPopupViewName to have it use the Enhanced Tooltip.

  Typically this is used to show a description of the image. That description often comes from another column in this table. Use the token "{COLUMN:column name}" to get text from a column whose name is part of the token.

  When you use the Expanded View feature, it helps to include text here that gives users instructions to click to view a larger size image.

  For example:

  ```
  ToolTipText="{COLUMN:ImageDesc}<br />Click to see a full size image."
  ```

  It defaults to "".

- **ToolTipTextLookupID** (string) – An alternative to **ToolTipText**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) Strings must be in the string group Misc. It defaults to "".

- **ToolTipPopupViewName** (string) – This determines the PopupView used for an Enhanced ToolTip. Specify the name or use "{DEFAULT}" to select the name defined globally in the **DefaultToolTipPopupViewName** property of the Global Settings Editor.

  If "", the standard browser tooltip is used.

  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.

  Details on PopupViews and Enhanced ToolTips can be found in the **Interactive Pages User's Guide**.

  It defaults to "{DEFAULT}".

Properties for a Label

See "Setting Properties on the Field Templates".

- **Label** (System.Web.UI.WebControls.Label) – Gets a reference to the Label control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `Label-propertyname='value'`. For example:

  ```
  <des:DynamicField DataField="ImageDesc"
   ReadOnlyPropertyList="Label-CssClass='myLabelCss'" />
  ```

- **LabelText** (string) – Text shown in the Label control. It can be HTML.

  Typically this is used to show a description of the image. That description often comes from another column in this table. Use the token "{COLUMN:column name}" to get text from a column whose name is part of the token.

  It defaults to "".

- **LabelTextLookupID** (string) – An alternative to **LabelText**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) Strings must be in the string group Misc. It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties for an Expanded View

See "Setting Properties on the Field Templates".

- **ExpandedViewUserControlUrl** (string) – The URL to a User Control that displays the image in an "expanded view" (usually full size). When assigned, the user can click on the image to display the Expanded View.

  When "", a click on the image does nothing.

  It defaults to "".

  Here are ExpandedView usercontrols shipped with DES Dynamic Data:

  `"~/DynamicData/Ajax Control Toolkit/ExpandedViewForImages.ascx"` - Provides a modal dialog designed for images using the Ajax Control Toolkit's ModalPopupExtender. Only available if you have installed support for the Ajax Control Toolkit. (To install, run the Web Application Updater with the "Update a web application" option. When it gets to the Third Party Controls panel, make sure you select the Ajax Control Toolkit.)

  If you develop an ExpandedView UserControl, it must implement the `PeterBlum.DES.DynamicData.Futures.IExpandedViewUI` interface. See "Creating Expanded View UserControls".

  The DES.DbImageDataTypeAttribute also has a **ExpandedViewUserControlUrl** property. It is only used when this property is unassigned.

- **TitleInExpandedView** (string) – Text shown in the Expanded View to annotate the image. It gets assigned to a Label control, as determined by the design of the Expanded View User Control. For example, **ExpandedViewForImages.ascx** uses it in the titlebar.

  Typically this is used to show a description of the image. That description often comes from another column in this table. Use the token "{COLUMN:column name}" to get text from a column whose name is part of the token.

  It defaults to "".

- **TitleInExpandedViewLookupID** (string) – An alternative to **TitleInExpandedView**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) Strings must be in the string group Misc. It defaults to "".

- **ExpandedViewImageMaxWidth** (integer) – The maximum width in pixels for the image shown in the Expanded View. If 0, it always uses the size of the image show. Otherwise, when the image is larger than this size, it is scaled to fit into this size.

  It defaults to 1000 (pixels).

- **ExpandedViewImageMaxHeight** (integer) – The maximum height in pixels for the image shown in the Expanded View. If 0, it always uses the size of the image show. Otherwise, when the image is larger than this size, it is scaled to fit into this size.

  It defaults to 1000 (pixels).

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties for Editing the Image File

Only on `PeterBlum.DES.DynamicData.DbImageEditFTUC`.

The Field Template provides Edit and Clear buttons by default. If you prefer not to use them, remove them and update the call to `SetUpOtherControls()` to pass `null`/`Nothing` in their place.

See "Setting Properties on the Field Templates".

- **FileUpload** (System.Web.UI.WebControls.FileUpload) – Gets a reference to the FileUpload control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `FileUpload-propertyname='value'`. For example:

  ```
  <des:DynamicField DataField="CustomerPhoto"
   ReadOnlyPropertyList="FileUpload-CssClass='myUploadCss'" />
  ```

  Note: Third party FileUpload controls can be supported by creating a subclass of PeterBlum.DES.DynamicData.BaseFileUploadControlMapper and assigning an instance to the DbImageEditFTUC.FileUploadControlMapper property. Contact tech support for assistance.

- **DisplayImage** (System.Nullable<bool>) – Determines if the Image is shown. When `true`, it is shown.

  When `null`/`Nothing`, the DES.DbImageDataTypeAttribute's **DisplayInEdit** property is used. If that isn't present, the image is displayed.

  It defaults to `null`/`Nothing`.

- **VirusErrorMessage** (string) – ErrorMessage text used when Virus detection code detects a virus. Virus detection requires adding your own code that uses the API of your antivirus software through the VirusDetection.DetectVirus event handler. See "Adding Virus Detection to uploaded files".

  Supports the token "{FILENAME}" to show the filename retrieved.

  It defaults to "`Virus detected in the file [{FILENAME}]. Upload aborted.`".

  DES.DbImageDataTypeAttribute.**VirusErrorMessage** is only used when this property is the default text or is "".

- **VirusErrorMessageLookupID** (string) – Gets the value for **VirusErrorMessage** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of ErrorMessage. If no match is found OR this is blank, **VirusErrorMessage** 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.

  DES.DbImageDataTypeAttribute.**VirusErrorMessageLookupID** is only used when this property is "".

- **BadFormatErrorMessage** (string) – ErrorMessage text used when the file is not a valid image.

  Supports the token "{FILENAME}" to show the filename retrieved.

  It defaults to "`The file [{FILENAME}] is not the correct format. Upload aborted.`".

  DES.DbImageDataTypeAttribute.**BadFormatErrorMessage** is only used when this property is the default text or is "".

- **BadFormatErrorMessageLookupID** (string) – Gets the value for **BadFormatErrorMessage** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of ErrorMessage. If no match is found OR this is blank, **BadFormatErrorMessage** 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.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

DES.DbImageDataTypeAttribute.**BadFormatErrorMessageLookupID** is only used when this property is "".

- **RequiredErrorMessage** (string) – ErrorMessage text assigned when the column requires a value but does not have one. If not assigned, it falls back to the RequiredAttribute value.

  It defaults to "`The field {LABEL} must be assigned a file.`".

  DES.DbImageDataTypeAttribute.**RequiredErrorMessage** is only used when this property is the default text or is "".

- **RequiredErrorMessageLookupID** (string) – Gets the value for **RequiredErrorMessage** through the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) The LookupID and its value should be defined within the String Group of ErrorMessage. If no match is found OR this is blank, **RequiredErrorMessage** 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.

  DES.DbImageDataTypeAttribute.**RequiredErrorMessageLookupID** is only used when this property is "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Creating Expanded View UserControls

The Expanded View User Control displays the data from a column in a popped up view, such as in a separate browser window or absolutely positioned (like the Ajax Control Toolkit's Modal Popup Extender and DES's PopupView). It appears when the user clicks on a small representation of the data shown on the page. For example, an image may be in a reduced size on the page or if it's a Word document, an Word icon may be on the page.

The Expanded View User Control creates the entire user interface for the popup, including its own scripts to show and hide itself. DES Dynamic Data supplies a URL to the get data. Your javascript code applies that Url to the <img> tag, <iframe> tag, or browser window. DES will return the data with the correct MIME type (like "image/gif" or "text/html"). So make your User Control capable of displaying the correct type. An <img> tag can handle image formats. <iframe> and browser windows can handle application specific types like for MS Word and PDF.

The Expanded View User Control must implement `PeterBlum.DES.DynamicData.IExpandedViewUI`. Here is that interface definition:

[C#]

```
public interface IExpandedViewUI
{
   string GetUserInterfaceID();

   void AddProperties(ScriptBuilder pScript, ExpandedViewUIArgs pArgs);

}
```

[VB]

```
Public Interface IExpandedViewUI
    Function GetUserInterfaceID() As String
    Sub AddProperties(ByVal pScript As ScriptBuilder,
         ByVal pArgs As ExpandedViewUIArgs)
End Interface
```

Please see the file **[DES product folder]/DynamicData/Ajax Control Toolkit/[language]/FieldTemplates/ExpandedViewForImages.ascx** for guidance on this interface.

The `PeterBlum.DES.DynamicData.ExpandedViewUIArgs` class has these properties:

- **DisplayDataControl** (System.Web.UI.Control) – The control that will show the data. When showing an image, it is usually an Image control. When showing an application specific type, like MS Word or PDF, it should either be an <iframe runat="server"> or not assigned. (When not assigned, your javascript will create a place for the URL, such as opening a separate window.) It is optional.
- **TitleLabelControl** (System.Web.UI.Control) – The control that will show the text that titles the data. It must allow updates through the innerHTML. It is optional.
- **LoadingImageUrl** (string) – URL to an image file that is shown while the desired image is still loading. Suggestion: "{APPEARANCE}/Shared/Spacer.gif" for a blank image. If you have an animated image that indicates loading, consider using it.
- **UpdateUIMethod** (string) – Method name. Required. This client-side method is called to display your user interface. It takes two parameters, EVUIO and EVO.

  For ModalPopupExtender, you can use "DES_DDEVUpdMP" which is already defined and loaded.

  The EVUIO parameter is a client-side object that describes the user interface. It has these properties (case sensitive):

  - **ID** - string. A unique ID for the user interface. The EVO.UIID property maps to this. When using a ModalPopupExtender, this is the BehaviorID.
  - **DataID** - string. The ClientID of the control declared in ExpandedViewUIArgs.DisplayDataControl. If null, you have to write code in this method that uses the EVO.Url property to assign a URL to your <img>, <iframe>, or browser window.
  - **LblID** - string. The ClientID of the control declared in ExpandedViewUIArgs.TitleLabelControl. It will be assigned data from EVO.Lbl. You do not have to write code for this.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **LUrl** - string. Url to an image that appears while loading the image. From ExpandedViewUIArgs.LoadingImageUrl.

The EVO parameter is a client-side object that describes the data from the record that your Expanded View shows. It has these properties (case sensitive):

- **ID** - string. The clientID of the Field Template that declared it.
- **UIID** - string. The ID to the EVUIO object that supplies the user interface.
- **Url** - string. The URL to assign to the image or file.
- **Lbl** - string. Text to show in the ExpandedView, such as in a title area.

Here are two UpdateUIMethods that are already defined and loaded. You may be able to use them.

```
// The Update method for anything that has a EVUIO.DataID declared.
// It updates the src property on that control.
// It both assigns the Url and shows the modal popup
function DES_DDEVUpdStd(pEVUIO, pEVO)
{
   var vFld = DES_GetById(pEVUIO.DataID);
   if (pEVUIO.LUrl)  // this is preloaded so it will show up right away.
   {
      vFld.src = pEVUIO.LUrl;
      window.setTimeout("DES_GetById('" + vFld.id + "').src='" + pEVO.Url + "';", 10);
   }
   else
      vFld.src = pEVO.Url;  // this will load after a delay from the server, leaving LUrl visible for a
moment
}  // DES_DDEVUpdStd

// The Update method for Modal Popups.
// It both assigns the Url and shows the modal popup
function DES_DDEVUpdMP(pEVUIO, pEVO)
{
   DES_DDEVUpdStd(pEVUIO, pEVO);
   var vMP = $find(pEVUIO.ID); // ID maps to the BehaviorID on the ModalPopupExtender
   if (vMP != null)
   {
      if (pEVUIO.LblID)
         DES_SetInnerHTML(DES_GetById(pEVUIO.LblID), pEVO.Lbl);
      vMP.show();
   }
}  // DES_DDEVUpdMP
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Adding Virus Detection to uploaded files

You should protect your database against viruses by having the uploaded files scanned before their data is saved. DbImage_Edit.ascx (actually the PeterBlum.DES.DynamicData.DbImageEditFTUC class) runs validation that can call your own code to check for viruses. If your code detects a virus, a validation error is returned to the user and the data is not saved.

To add virus detection, you create a method with this definition. It may be helpful to declare it in the **Global.asax** file as its called from code setup in `Application_Start()`.

[C#]

```
public bool IsVirusFree(PeterBlum.DES.DynamicData.FileContentArgs pArgs)
{
  if (ReviewFileBytes(pArgs.FileBytes))   // you write the ReviewFileBytes code
    return true; // no viruses
  else
    return false;  // has a virus!
}
```

[VB]

```
Public Function IsVirusFree(ByVal pArgs As PeterBlum.DES.DynamicData.FileContentArgs)_
  As Boolean

  If ReviewFileBytes(pArgs.FileBytes) Then ' you write the ReviewFileBytes code
    Return True ' no viruses
  Else
    Return False   ' has a virus!
  End If
End Function
```

The pArgs parameter only contains one property, FileBytes, which is an array of bytes containing the entire file that was uploaded.

You must connect your `IsVirusFree()` method to the VirusDetection.DetectVirus event handler in `Application_Start()` like this:

[C#]

```
PeterBlum.DES.DynamicData.VirusDetection.DetectVirus +=
  new PeterBlum.DES.DynamicData.EvaluateFileContents(IsVirusFree);
```

[VB]

```
AddHandler PeterBlum.DES.DynamicData.VirusDetection.DetectVirus _
  AddressOf IsVirusFree
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

UrlAsImage.ascx and UrlAsImage_Edit.ascx Field Templates

Field Template Base classes	PeterBlum.DES.DynamicData.UrlAsImageStaticFTUC PeterBlum.DES.DynamicData.UrlAsImageEditFTUC
Field Template files	UrlAsImage.ascx, UrlAsImage_Edit.ascx
DataTypes supported	String
DataControl types	Image Label TextBox (for Edit/Insert Template)

This Field Template uses an Image control to display a Url pointing to an image. Usually you will establish maximum width and height limits to fit the image on the page. It can optionally display an "Expanded View", which shows the same image in its actual size. The Expanded View opens when the user clicks on the image.

You setup the Expanded View by assigning a Url to a specialized UserControl within the **ExpandedViewUserControlUrl** property. DES Dynamic Data includes an Expanded View that uses the Ajax Control Toolkit's Modal Popup Extender in "~/DynamicData/FieldTemplates/Ajax Control Toolkit". For more on creating other Expanded View user controls, see "Creating Expanded View UserControls".

This Field Template optionally shows text in a Label control to annotate the image. The text shown is assigned in the **LabelText** property.

When in Edit and Insert mode, `PeterBlum.DES.DynamicData.UrlAsImageEditFTUC` provides a TextBox to edit the Url. It's validation checks that the file has the file extendion of one of the expected types (determined by DES.UrlAsImageDataTypeAttribute.SupportedFileTypes). It also provides client-side scripts that update the image shown after an edit is made.

Click on any of these topics to jump to them:

- Properties on the Image
- Properties for a ToolTip
- Properties for a Label
- Properties for an Expanded View
- Properties for Editing the Url
- Properties Common to All Field Templates
- Properties on the Customizer Control
- Creating Expanded View UserControls

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties on the Image

See "Setting Properties on the Field Templates".

- **Image** (System.Web.UI.WebControls.Image) – Gets a reference to the Image control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `Image-propertyname='value'`. For example:

```
<des:DynamicField DataField="CustomerPhoto"
 ReadOnlyPropertyList="Image-Width='80px'" />
```

- **UrlFormat** (string) – The text to assign to the Image's **ImageUrl** property on the Field Template. Use the token "{FIELD}" to insert the text from the column. If there is no token, it appends the text from the column. Use this when your stored Url is missing some elements of the path, like the domain name. The format will contain those missing elements.

  The DES.UrlAsImageDataTypeAttribute can also specify this value using its **UrlFormat** property. The value from **DES.UrlAsImageDataTypeAttribute.UrlFormat** is only used when this property is "".

  It defaults to "".

- **ImageWidth** (integer) – Overrides the image's actual width with this value, in pixels. When 0, use the image's actual width. The DES.UrlAsImageDataTypeAttribute can also specify a width using its **DisplayWidth** property. The value from **DES.UrlAsImageDataTypeAttribute.DisplayWidth** is only used when this property is 0.

  It defaults to 0.

- **ImageHeight** (integer) – Overrides the image's actual height with this value, in pixels. When 0, use the image's actual height. The DES.UrlAsImageDataTypeAttribute can also specify a height using its **DisplayHeight** property. The value from **DES.UrlAsImageDataTypeAttribute.DisplayHeight** is only used when this property is 0.

  It defaults to 0.

- **MissingImageUrl** (string) – An image to show when the database does not supply an image. If not assigned when needed, the Image control is hidden.

  It defaults to `"{APPEARANCE}/Shared/Spacer.gif"` which is a transparent 1x1 gif.

- **ImageAlternativeText** (string) – The alternative text used by the image on the browser. The browser will display it as a tooltip (unless overridden by **ToolTipText**) and on the page when the actual image Url doesn't return an image. It defaults to "".

- **ImageAlternativeTextLookupID** (string) – An alternative to **ImageAlternativeText**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) Strings must be in the string group Misc. It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties for a ToolTip

See "Setting Properties on the Field Templates".

- **ToolTipText** (string) – Text shown in the tooltip. The tooltip can be an Enhanced Tooltip (a feature of the Peter's Interactive Pages module that uses a PopupView to display an HTML-driven tooltip.) You must set ToolTipPopupViewName to have it use the Enhanced Tooltip.

  Typically this is used to show a description of the image. That description often comes from another column in this table. Use the token "{COLUMN:column name}" to get text from a column whose name is part of the token.

  When you use the Expanded View feature, it helps to include text here that gives users instructions to click to view a larger size image.

  For example:

  ```
  ToolTipText="{COLUMN:PhotoDesc}<br />Click to see a full size image."
  ```

  It defaults to "".

- **ToolTipTextLookupID** (string) – An alternative to **ToolTipText**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) Strings must be in the string group Misc. It defaults to "".

- **ToolTipPopupViewName** (string) – This determines the PopupView used for an Enhanced ToolTip. Specify the name or use "{DEFAULT}" to select the name defined globally in the **DefaultToolTipPopupViewName** property of the Global Settings Editor.

  If "", the standard browser tooltip is used.

  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.

  Details on PopupViews and Enhanced ToolTips can be found in the **Interactive Pages User's Guide**.

  It defaults to "{DEFAULT}".

Properties for a Label

See "Setting Properties on the Field Templates".

- **Label** (System.Web.UI.WebControls.Label) – Gets a reference to the Label control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `Label-propertyname='value'`. For example:

  ```
  <des:DynamicField DataField="PhotoDesc"
   ReadOnlyPropertyList="Label-CssClass='myLabelCss'" />
  ```

- **LabelText** (string) – Text shown in the Label control. It can be HTML.

  Typically this is used to show a description of the image. That description often comes from another column in this table. Use the token "{COLUMN:column name}" to get text from a column whose name is part of the token.

  It defaults to "".

- **LabelTextLookupID** (string) – An alternative to **LabelText**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) Strings must be in the string group Misc. It defaults to "".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties for an Expanded View

See "Setting Properties on the Field Templates".

- **ExpandedViewUserControlUrl** (string) – The URL to a User Control that displays the image in an "expanded view" (usually full size). When assigned, the user can click on the image to display the Expanded View.

  When "", a click on the image does nothing.

  It defaults to "".

  Here are ExpandedView usercontrols shipped with DES Dynamic Data:

  `"~/DynamicData/Ajax Control Toolkit/ExpandedViewForImages.ascx"` - Provides a modal dialog designed for images using the Ajax Control Toolkit's ModalPopupExtender. Only available if you have installed support for the Ajax Control Toolkit. (To install, run the Web Application Updater with the "Update a web application" option. When it gets to the Third Party Controls panel, make sure you select the Ajax Control Toolkit.)

  If you develop an ExpandedView UserControl, it must implement the `PeterBlum.DES.DynamicData.Futures.IExpandedViewUI` interface. See "Creating Expanded View UserControls".

  The DES.UrlAsImageDataTypeAttribute also has a **ExpandedViewUserControlUrl** property. It is only used when this property is unassigned.

- **TitleInExpandedView** (string) – Text shown in the Expanded View to annotate the image. It gets assigned to a Label control, as determined by the design of the Expanded View User Control. For example, **ExpandedViewForImages.ascx** uses it in the titlebar.

  Typically this is used to show a description of the image. That description often comes from another column in this table. Use the token "{COLUMN:column name}" to get text from a column whose name is part of the token.

  It defaults to "".

- **TitleInExpandedViewLookupID** (string) – An alternative to **TitleInExpandedView**. When used, the error message is looked up from another source, such as a resource file or database using the String Lookup System. (See "String Lookup System" in the **General Features Guide**.) Strings must be in the string group Misc. It defaults to "".

Properties for Editing the Url

See "Setting Properties on the Field Templates".

- **TextBox** (System.Web.UI.WebControls.TextBox) – Gets a reference to the TextBox control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `TextBox-propertyname='value'`. For example:

```
<des:DynamicField DataField="UrlToPhoto"
 ReadOnlyPropertyList="TextBox-CssClass='myTextBoxCss'" />
```

- **Columns** (integer) – Sets the **Columns** property on the TextBox. When 0, it does not modify the TextBox.Columns property, allowing you to set it within the Field Template. It defaults to 0.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Url.ascx Field Template

Field Template Base classes	PeterBlum.DES.DynamicData.UrlStaticFTUC
Field Template files	Url.ascx
DataTypes supported	String
DataControl types	HyperLink

Properties

See "Setting Properties on the Field Templates".

- **HyperLink** (System.Web.UI.WebControls.HyperLink) – Gets a reference to the HyperLink control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `HyperLink-propertyname='value'`. For example:

```
<desDD:DynamicField DataField="UrlToLogo"
 ReadOnlyPropertyList="HyperLink-CssClass='myHyperLinkCss'" />
```

- **UriScheme** (string) – If the URL lacks a UriScheme (http, https, etc), this is what to use. It is inserted at the beginning of the Url.

  Do not include the "://" element.

  It defaults to "`http`".

- **Target** (string) – Determines the value assigned to the HyperLink's Target property. It defaults to "`_blank`".

- **UrlText** (string) – Determines the value assigned to the HyperLink's Text property, allowing you to customize the text displayed to the user.

  Supports these tokens:

  {URL} - The full URL. This is the default.

  {FILENAME} - The filename element of the URL.

  {DOMAIN} - The domain element of the URL.

  It defaults to "`{URL}`".

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Url_Edit.ascx Field Template

Field Template Base classes	PeterBlum.DES.DynamicData.UrlEditFTUC
Field Template files	Url_Edit.ascx
DataTypes supported	String
DataControl types	TextBox

This Field Template inherits from the same base class as **Text_Edit.ascx**.

Properties

See "Text_Edit.ascx and MultilineText_Edit.ascx Field Templates".

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

EmailAddress.ascx Field Template

Field Template Base classes	PeterBlum.DES.DynamicData.EmailAddressStaticFTUC
Field Template files	EmailAddress.ascx
DataTypes supported	String
DataControl types	Label, Panel, PlaceHolder (will be assigned a LiteralControl with content for you), and HyperLink (only supports the first email address when multiple are defined).

Properties

See "Setting Properties on the Field Templates".

- **HyperLink** (System.Web.UI.WebControls.HyperLink) – Gets a reference to the HyperLink control registered on this Field Template. Use it to set properties directly on this control when you are using a HyperLink. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `HyperLink-propertyname='value'`. For example:

```
<desDD:DynamicField DataField="UrlToLogo"
 ReadOnlyPropertyList="HyperLink-CssClass='myHyperLinkCss'" />
```

- **ShowAsHyperlink** (enum PeterBlum.DES.TrueFalseDefault) – The email addresses can be displayed in either static or hyperlink (mailto://) format (except when using a Hyperlink control).

  When `TrueFalseDefault.True`, display as hyperlink.

  When `TrueFalseDefault.False`, display as static text.

  When `TrueFalseDefault.Default`, get the value from the **Customizer.EmailAddress_ShowAsHyperlink** property. This is the default.

  It has no effect when the DataControl is a Hyperlink control.

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

EmailAddress_Edit.ascx Field Template

Field Template Base classes	PeterBlum.DES.DynamicData.EmailAddressEditFTUC
Field Template files	EmailAddress_Edit.ascx
DataTypes supported	String
DataControl types	TextBox

Properties

See "Setting Properties on the Field Templates".

- **ShowExampleInHint** (enum PeterBlum.DES.TrueFalseDefault) – The EmailAddressDataTypeAttribute generates an example to be shown when there is a validation error. It can also appear in a hint as the user moves into the textbox.

  When `TrueFalseDefault.True`, show a hint with the example.

  When `TrueFalseDefault.False`, do not show a hint with the example.

  When `TrueFalseDefault.Default`, get the value from the **Customizer. FormattedString_ShowExampleInHint** property. This is the default.

  It has no effect when the DataControl is a Hyperlink control.

See also "Text_Edit.ascx and MultilineText_Edit.ascx Field Templates" as this Field Template uses the same properties.

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Calendar_Edit.ascx Field Template

Field Template Base classes	PeterBlum.DES.DynamicData.CalendarEditFTUC
Field Template files	Calendar_Edit.ascx
DataTypes supported	DateTime
DataControl types	PeterBlum.DES.Calendar

Properties

See "Setting Properties on the Field Templates".

- **Calendar** (PeterBlum.DES.Calendar) – Gets a reference to the Calendar control registered on this Field Template. Use it to set properties directly on this control. When using DES.UIHintAttribute or DES.DynamicField, they allow this syntax within their *xyz*PropertyList properties: `Calendar-propertyname='value'`. For example:

```
<des:DynamicField DataField="BirthDate"
 EditPropertyList="Calendar-MultiMonthColumnCount='2'" />
```

See also "Properties Common to All Field Templates" and "Properties on the Customizer Control".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using Style Sheets within Field Templates

DES provides style sheet classes for these Field Templates within the **[web app]/DynamicData/Contents/FieldTemplates.css** file. This file is automatically loaded for you when your Field Template sets the **AutoLoadFieldTemplateStyleSheetFile** property to `true`. See below. Customize the styles as you need, or modify the Field Template file to use other style sheet classes.

When used, it merges the style sheet file with other DES files. It also compresses the file, removing comments and much whitespace.

When not used, you can still load it merged and compressed by calling

```
PeterBlum.DES.Globals.Page.StyleSheetManager.Include(
       PeterBlum.DES.StyleSheetFiles.Field Templates)
```

Adding Style Sheet Classes to your Field Templates

1. Define style sheet classes in the **[web app]/DynamicData/Contents/FieldTemplates.css** file. *You are not required to use this file. If you have another, you may need to manually load it or change the default path to your file. See below.*
2. Assign the classes to your Field Template's controls and HTML.
3. Activate automatic loading of the Field Templates.css file by setting the **AutoLoadFieldTemplateStyleSheetFile** property to `true` in the `Page_Init()` method.

   Example

   [C#]

```
protected void Page_Init(object sender, EventArgs e)
{
   SetUpStaticDataControl(controlid);

   AutoLoadFieldTemplateStyleSheetFile = true;
}
```

   [VB]

```
Protected Sub Page_Init(ByVal sender As Object, ByVal e As EventArgs)
   SetUpStaticDataControl(controlid)

   AutoLoadFieldTemplateStyleSheetFile = True
End Sub
```

Changing the default path to the Field Templates Style Sheet File

The path to this file can be changed in two ways:

- In `Application_Start()`, call

```
PeterBlum.DES.StyleSheetManager.SetDefaultUrl(
 PeterBlum.DES.StyleSheetFiles.Field Templates, "URL")
```

- In the `<appSettings>` section of **web.config**, add

```
<add key="DES_StyleSheetFieldTemplates" value="URL" />
```

It defaults to `true`.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Customization based on Categories

Use the CategoryAttribute within your Entity class metadata to categorize a column. The Field Template can use this information to change its appearance. Suppose that you want certain columns to be known as "Titles" and that results in the user interface changing from the standard style sheet class to one that makes a bigger font. Assign the CategoryAttribute to those columns and build logic in the Field Template's `Page_Load()` method that uses the `CheckCategories()` method, declared on the DESFieldTemplateUserControl class.

[C#]

```
public class CategoryMetadata
{
   [Category("Title")]
   public object CategoryName { get; set; }
}
```

In the Field Template:

```
<script runat="server">
   protected void Page_Init(object sender, EventArgs e)
   {
      SetUpStaticDataControl(FieldValueLabel);
   }
   protected void Page_Load(object sender, EventArgs e)
   {
      if (CheckCategories("Title", false, false))
         FieldValueLabel.CssClass = "TitleLabel";
      else
         FieldValueLabel.CssClass = "StndLabel";
   }
</script>
<asp:Label ID="FieldValueLabel" runat="server" ></asp:Label>
```

[VB]

```
Public Class CategoryMetadata
   <Category("Title")> _
   Public Property CategoryName As Object
      Get
         Return Nothing
      End Get
      Set
      End Set
   End Property
End Class
```

In the Field Template:

```
<script runat="server">
   Protected Sub Page_Init(ByVal sender As Object, ByVal e As EventArgs)
      SetUpStaticDataControl(FieldValueLabel)
   End Sub
   Protected Sub Page_Load(ByVal sender As Object, ByVal e As EventArgs)
      If CheckCategories("Title", False, False) Then
         FieldValueLabel.CssClass = "TitleLabel"
      Else
         FieldValueLabel.CssClass = "StndLabel"
      End If
   End Sub
</script>
<asp:Label ID="FieldValueLabel" runat="server" ></asp:Label>
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

DESFieldTemplateUserControl.CheckCategories method

Checks the column's CategoryAttribute against the filtering rules to determine if there is a match. It is safe to call even when there is no CategoryAttribute on the column's metadata.

[C#]

```
public bool CheckCategories(string pCategories)
```

```
public bool CheckCategories(string pCategories, bool pExcludeCategories,
  bool pCategoriesWithNoMatchAreExcluded)
```

[C#]

```
Public Function CheckCategories(ByVal pCategories As String,_
  ByVal pExcludeCategories As Boolean, _
  ByVal pCategoriesWithNoMatchAreExcluded As Boolean) As Boolean
```

Parameters

pCategories

A pipe delimited list of Categories to match against the **CategoryAttribute.Value** property.

pExcludeCategories

When `false`, at least one of the category names in *pCategories* must be in the CategoryAttribute.

When `true`, none of the category names in *pCategories* can be found be in the CategoryAttribute.

If this parameter is omitted, it is `false` (categories must match.)

pCategoriesWithNoMatchAreExcluded

Determines what happens when *pCategories* does not contain a matching name to the CategoryAttribute's value, including when the Columns' metadata lacks a CategoryAttribute.

When `false`, return `false`.

When `true`, return `true`.

If this parameter is omitted, this is `false` (return `false`).

Returns

When `true`, the input parameters were matched. When `false`, they were not.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Implementing the ICustomizeFieldTemplate Interface

The `PeterBlum.DES.DynamicData.ICustomizeFieldTemplate` Interface lets you modify properties on a Field Template at runtime. It is called after all webcontrols have been created and registered with the Field Template, allowing you to directly modify properties on those controls. For example, you can edit the IntegerTextBox of Integer_Edit.ascx to set its BackColor property.

Each FieldTemplateUserControl class has public properties to expose key controls.

- DataControl – The control showing and possibly editing the data. Many subclasses of DESFieldTemplateUserControl have properties that offer this value correctly typecasted to the actual control. For example, Text_Edit.ascx exposes its FilteredTextBox with a property called TextBox and Date_Edit.aspx exposes its DateTextBox with a property called DateTextBox.
- ColumnValidatorManager – The control containing the validators. Use its FindValidator method to get a validator.

As a result, you can avoid using the `FindControl()` method and knowing the control ID of a control on the Field Template.

The `PeterBlum.DES.DynamicData.ICustomizeFieldTemplate` Interface can be assigned to almost any object, although typically you put it on a UserControl, Page, or MasterPage class.

Once assigned, create the `CustomizeFieldTemplate()` method. It is called for each Field Template on the page. Its Field Template parameter is a reference to the Field Template that you want to modify. Use the `IsMatch()` methods on its Args parameter to detect if this is the desired Field Template. The Args parameter also has numerous properties that typecast and expose the DataControl property for easy access to its properties.

Click on any of these topics to jump to them:

- Where to implement ICustomizeFieldTemplate
- Adding the ICustomizeFieldTemplate Interface
- CustomizeFieldTemplate method
  - Properties of the CustomizeFieldTemplateEventArgs Class
  - Methods of the CustomizeFieldTemplateEventArgs Class

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Where to implement ICustomizeFieldTemplate

As an interface, you attach it to a class then create the list of methods within that class. You already have some classes created where you should consider attaching it: the Page, the MasterPage, or any UserControl containing the DynamicDataManager control. If you use any of these, DES will find and use it. You can also explicitly assign the class to the Customizer control's **CustomizeFieldTemplateOwner** property.

How DES locates your interface

You may have this interface declared in a few classes, such as both the MasterPage and UserControl containing a DynamicDataManager. If you let DES search for the interface, it always searches in this order:

1. User control with DynamicDataManager
2. Master Page
3. Page

How to specify which object to use

If you need to specify which of these to use, assign the object to the Customizer control's **CustomizeFieldTemplateOwner** property.

Always set the **CustomizeFieldTemplateOwner** property in the DynamicDataManager's Init event handler method or `Page_Init()` method.

[C#]

```
Customizer1.CustomizeFieldTemplateOwner = this;
```

[VB]

```
Customizer1.CustomizeFieldTemplateOwner = Me
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Adding the ICustomizeFieldTemplate Interface

1. Start by using the `PeterBlum.DES.DynamicData` namespace. It's easiest to map it to the name "DES" just like was done in the code file containing the EntityTable Metadata Classes.

   [C#]

   ```
   using DES = PeterBlum.DES.DynamicData;
   ```

   [VB]

   ```
   Import DES = PeterBlum.DES.DynamicData
   ```

2. Assign the ICustomizeFieldTemplate interface to the desired class.

   [C#]

   ```
   public class WebForm1 : System.Web.UI.WebControls.Page,
       DES.ICustomizeFieldTemplate
   ```

   [VB]

   ```
   Public Class WebForm1
      Implements System.Web.UI.WebControls.Page, _
      Implements DES.ICustomizeFieldTemplate
   ```

3. Implement the `CustomizeFieldTemplate()` method. See "CustomizeFieldTemplate method".

4. If its not already on the page, add the Customizer control. It is the conduit between your method and the code that invokes it. When using Page Templates, typically the Customizer is added to the MasterPage.

5. DES Dynamic Data will discover your control or page class that implements ICustomizeFieldTemplate so long as that class is a User Control that contains the DES DynamicDataManager control, the MasterPage, or the Page. See "How DES locates your interface".

   If it cannot be discovered, assign the object to the Customizer's **CustomizeFieldTemplateOwner** property in either the DynamicDataManager's Init event handler or the `Page_Init()` method like this:

   [C#]

   ```
   Customizer1.CustomizeFieldTemplateOwner = this;
   ```

   [VB]

   ```
   Customizer1.CustomizeFieldTemplateOwner = Me
   ```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

CustomizeFieldTemplate method

`CustomizeFieldTemplate()` is called for each Field Template on the page. (It only returns DES's Field Templates, based on the DESFieldTemplateUserControl class.) Its Field Template parameter is a reference to the Field Template that you want to modify. Use the `IsMatch()` methods on its Args parameter to detect if this is the desired Field Template. The Args parameter also has numerous properties that typecast and expose the DataControl property for easy access to its properties.

[C#]

```
public void CustomizeFieldTemplate(DES.DESFieldTemplateUserControl pFieldTemplate,
   CustomizeFieldTemplateEventArgs pArgs)
```

[VB]

```
Public Sub CustomizeFieldTemplate( _
   ByVal pFieldTemplate As DES.DESFieldTemplateUserControl, _
   ByVal pArgs As CustomizeFieldTemplateEventArgs)
```

Parameters

pFieldTemplate

The FieldTemplateUserControl to modify. Often you will only use the properties on the pArgs parameter to access its controls.

pArgs

An object that helps you determine if this is the desired Field Template in its `IsMatch()` method and provides numerous properties to expose controls found on the Field Template. See "Properties of the CustomizeFieldTemplateEventArgs Class" and "Methods of the CustomizeFieldTemplateEventArgs Class".

Generally you need to compare the Field Template to three values before you can determine it's the desired one. Those values are the TableName, ColumnName, and Mode (ReadOnly, Edit, and Insert). `IsMatch()`takes those values as parameters. Use `IsMatchByDataControlType()` to detect a specific type of data control that you want to edit throughout all Field Templates your method is passed. Use `IsMatchByFileName()` to determine if the Field Template is a specific filename.

It exposes different typecasts of the FieldTemplateUserControl.DataControl property using the syntax: "DataControlAs" + Type. For example, DataControlAsLabel and DataControlAsFilteredTextBox.

Example

Demonstrates all three IsMatch methods. (See "Methods of the CustomizeFieldTemplateEventArgs Class".)

Changes the background color for the ContactName column of the Customers table. Changes the border color of all FilteredTextBoxes. Changes the Columns property of textboxes on the Multiline_Edit.aspx Field Template. *Note: The Customizer control can set the Columns property too.*

[C#]

```
public void CustomizeFieldTemplate(
   DES.DESFieldTemplateUserControl pFieldTemplate,
   DES.CustomizeFieldTemplateEventArgs pArgs)
{
   if (pArgs.IsMatch("Customers", "ContactName", DataBoundControlMode.ReadOnly))
      pArgs.DataControlAsLabel.BackColor = System.Drawing.Color.Cornsilk;

   if (pArgs.IsMatchByType(typeof(PeterBlum.DES.FilteredTextBox), false))
      pArgs.DataControlAsFilteredTextBox.BorderColor = System.Drawing.Color.DarkOrange;

   if (pArgs.IsMatchFieldTemplateName("MultilineText_Edit.ascx"))
      pArgs.DataControlAsTextBox.Columns = 60;
}
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

[VB]

```
Public Sub CustomizeFieldTemplate( _
   ByVal pFieldTemplate As DES.DESFieldTemplateUserControl, _
   ByVal pArgs As DES.CustomizeFieldTemplateEventArgs) _
  Implements DES.ICustomizeFieldTemplate.CustomizeFieldTemplate

   If pArgs.IsMatch("Customers", "ContactName", DataBoundControlMode.ReadOnly) Then
      pArgs.DataControlAsLabel.BackColor = System.Drawing.Color.Cornsilk
   End If
   If pArgs.IsMatchByType(GetType(PeterBlum.DES.FilteredTextBox), False) Then
      pArgs.DataControlAsFilteredTextBox.BorderColor = System.Drawing.Color.DarkOrange
   End If
   If pArgs.IsMatchFieldTemplateName("MultilineText_Edit.ascx") Then
      pArgs.DataControlAsTextBox.Columns = 60
   End If
End Sub
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Properties of the CustomizeFieldTemplateEventArgs Class

- **Column** (MetaColumn) – The MetaColumn for this Field Template.
- **TableName** (string) – The table name for this Field Template.
- **ColumnName** (string) – The column name.
- **Mode** (enum System.Web.UI.WebControls.DataBoundControlMode) – The mode of the Field Template: `ReadOnly`, `Edit`, or `Insert`.
- **FileName** (string) – The filename of the Field Template. It does not include the path. It does include the file extension. Consider using IsMatchByFileName() to do a case insensitive compare to this value.
- **DataControl** (Control) – The **DataControl** property on the Field Template. It has a reference to the control that displays and edits the data. The remaining properties are typecasts of this property. If you don't see the right type below, typecast this property.
- **DataControlAsTextBox** – Typecast of **DataControl** to `System.Web.UI.WebControls.TextBox`. `Null`/`Nothing` when its not a TextBox.
- **DataControlAsDESTextBox** – Typecast of **DataControl** to `PeterBlum.DES.TextBox`. `Null`/`Nothing` when its not a DES TextBox.
- **DataControlAsFilteredTextBox** – Typecast of **DataControl** to FilteredTextBox. `Null`/`Nothing` when its not a FilteredTextBox. **Text_Edit.ascx** uses this control. (**MultilineText_Edit.ascx** does not. It uses a DES TextBox).
- **DataControlAsLabel** – Typecast of **DataControl** to `System.Web.UI.WebControls.Label`. `Null`/`Nothing` when its not a Label.
- **DataControlAsListControl** – Typecast of **DataControl** to `System.Web.UI.WebControls.ListControl` which is the ancestor to ListBox, DropDownList, and others. `Null`/`Nothing` when its not a ListControl.
- **DataControlAsCheckBox** – Typecast of **DataControl** to `System.Web.UI.WebControls.CheckBox`. It also supports the RadioButton control. `Null`/`Nothing` when its not a CheckBox or RadioButton.
- **DataControlAsHyperLink** – Typecast of **DataControl** to `System.Web.UI.WebControls.HyperLink`. `Null`/`Nothing` when its not a HyperLink.
- **DataControlAsDateTextBox** – Typecast of **DataControl** to DateTextBox. `Null`/`Nothing` when its not a DateTextBox.
- **DataControlAsTimeOfDayTextBox** – Typecast of **DataControl** to TimeOfDayTextBox. `Null`/`Nothing` when its not a TimeOfDayTextBox.
- **DataControlAsTimeOfDayTextBox** – Typecast of **DataControl** to DurationTextBox. `Null`/`Nothing` when its not a DurationTextBox.
- **DataControlAsIntegerTextBox** – Typecast of **DataControl** to IntegerTextBox. `Null`/`Nothing` when its not a IntegerTextBox.
- **DataControlAsDecimalTextBox** – Typecast of **DataControl** to DecimalTextBox. `Null`/`Nothing` when its not a DecimalTextBox.
- **DataControlAsCurrencyTextBox** – Typecast of **DataControl** to CurrencyTextBox. `Null`/`Nothing` when its not a CurrencyTextBox.
- **DataControlAsPercentTextBox** – Typecast of **DataControl** to PercentTextBox. `Null`/`Nothing` when its not a PercentTextBox.
- **DataControlAsMonthYearTextBox** – Typecast of **DataControl** to MonthYearTextBox. `Null`/`Nothing` when its not a MonthYearTextBox.
- **DataControlAsAnniversaryTextBox** – Typecast of **DataControl** to AnniversaryTextBox. `Null`/`Nothing` when its not a AnniversaryTextBox.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

- **ColumnValidatorManager** – A reference to the ColumnValidatorManager control on the page. Use the `pArgs.FindValidator()` method to get a specific Validator. See “FindValidator Method”.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Methods of the CustomizeFieldTemplateEventArgs Class

Click on any of these topics to jump to them:

- IsMatch Method
- IsMatchByDataControlType Method
- IsMatchByFileName Method
- FindValidator Method

IsMatch Method

Match based on table name, column name and optionally the mode.

[C#]

```
public bool IsMatch(string pTableName, string pColumnName)
public bool IsMatch(string pTableName, string pColumnName, bool pReadOnly)
public bool IsMatch(string pTableName, string pColumnName,
  DataBoundControlMode pMode)
```

[VB]

```
Public Function IsMatch(ByVal pTableName As String, ByVal pColumnName As String) _
  As Boolean
Public Function IsMatch(ByVal pTableName As String, ByVal pColumnName As String, _
  ByVal pReadOnly As Boolean) As Boolean
Public Function IsMatch(ByVal pTableName As String, ByVal pColumnName As String, _
  ByVal pMode As DataBoundControlMode) As Boolean
```

Parameters

pTableName

The table name you want.

pColumnName

The column name you want.

pReadOnly

When `true`, the Mode must be ReadOnly. When `false`, the Mode must be Edit or Insert.

pMode

Alternative to *pReadOnly* that gives more control over which Mode you are looking for.

Returns

When `true`, the Field Template matches the parameters. When `false`, it does not.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

IsMatchByDataControlType Method

Match based on the class type of the DataControl property.

[C#]

```
public bool IsMatchByDataControlType(Type pDataControlType, bool pExact)
```

[VB]

```
Public Function IsMatchByDataControlType (ByVal pDataControlType As Type, _
  ByVal pExact As Boolean) As Boolean
```

Parameters

pType

The class type of the DataControl property that you are looking for.

pExact

When `true`, match to the exact type. When `false`, match to the exact type and any subclass of that type.

Returns

When `true`, the Field Template matches the parameters. When `false`, it does not.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

IsMatchByFileName Method

Match to the file name of the Field Template. It can also evaluate the folders containing the file by passing a partial path leading up to the file name. Uses a case insensitive comparison.

[C#]

```
public bool IsMatchByFileName(string pFieldTemplateName)
```

[VB]

```
Public Function IsMatchByDataControlType (ByVal pFieldTemplateName As String) _
  As Boolean
```

Parameters

pFieldTemplateName

The file name. The ".ascx" extension is optional. If you want to match to containing folders, supply them. For example, "DES/FieldTemplates/Integer.ascx" will not match to a file in the "RadControls/FieldTemplates" folder.

Returns

When `true`, the Field Template matches the parameters. When `false`, it does not.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

FindValidator Method

Get a reference to a validator on the ColumnValidatorManager. Specify the validator's type. You can modify properties that are used in the user interface, like ErrorMessage and ErrorFormatter. You should not modify the data evaluation properties.

[C#]

```
public PeterBlum.DES.BaseAnyValidator FindValidator(Type pTypeOfValidator)
```

[VB]

```
Public Function FindValidator (ByVal pTypeOfValidator As Type) _
  As PeterBlum.DES.BaseAnyValidator
```

Parameters

pTypeOfValidator

The type of the validator to return.

Returns

The Validator object if found. `Null`/`Nothing` if not found. The class BaseAnyValidator has access to all of the properties shared amongst the DES validators. If you need to access ControlIDToEvaluate, SecondControlIDToEvaluate or any Label property, typecast to the specific validator type.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

FieldTemplateFactory: Selecting the correct Field Template file

ASP.NET Dynamic Data uses the System.Web.DynamicData.FieldTemplateFactory class to determine exactly which Field Template file is used. It does so by using the actual column's datatype, the DataTypeAttribute, the UIHintAttribute, the mode (readonly, edit, insert), and more. See "Understanding how Field Templates are selected".

DES replaces the native FieldTemplateFactory class with `PeterBlum.DES.DynamicData.DESFieldTemplateFactory` to enhance the ability to lookup Field Templates.

It's enhancments include:

- Use aliases for Field Templates. Effectively this is a global UIHintAttribute. For example, always map Date columns to Calendar_Edit.ascx in edit mode or all boolean columns to BooleanCheckBox.ascx in readonly mode.
- Determines which Field Template to use when there are several of the same name in different folders. This allows several third parties to deliver their own Field Templates that are for the same datatype, such as "Integer_Edit.ascx".
- Provides the CreateTemplate event where you can intercept any request and return an instance of a Field Template, whether its loaded from a UserControl file or created programmatically.
- Can work without the Field Templates folder by creating Field Template objects populated with a default set of controls.

The DESFieldTemplateFactory uses an XML file to setup its rules. The XML file is called **FieldTemplates.config** and is located in the **[web application]/DynamicData** folder. You can edit it with a text editor.

See "Steps to add DES Dynamic Data support" to establish the DESFieldTemplateFactory on your MetaModel.

Click on any of these topics to jump to them:

- FieldTemplates.config File Format
- Understanding how Field Templates are selected
- Creating Field Templates programmatically

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

FieldTemplates.config File Format

Note: Anytime this file is changed, be sure to restart your web application as Dynamic Data caches its contents.

The DESFieldTemplateFactory class uses the **[web application]/DynamicData/FieldTemplates.config** to customize how it selects a Field Template. You define special cases such as aliases and alternative locations in the **FieldTemplates.config** file. An alias allows you to globally map the Field Template name requested for a specific datatype to another, such as always use **Calendar_Edit.ascx** instead of **Date_Edit.ascx** on a Date column. An alternative location allows you to have several cases of the same named Field Template in different folders and map which will be actually loaded.

Here is the file format of this XML file:

```
<?xml version="1.0" encoding="UTF-8"?>
<FieldTemplateFactory>
  <aliases>
    <alias origname="name"
      namesource="DataType|UIHint|Any"
      mode="readonly|edit|insert"
      newname="name"
      enabled="true|false" />
  </aliases>
  <pathsearch>
    <path url="url" />
  </pathsearch>
  <templates>
    <template name="Field Template name"
      url="name"
      modes="readonly|edit|insert"
      enabled="true|false" />
  </templates>
</FieldTemplateFactory>
```

Click on any of these topics to jump to them:

- Aliases Section
- Pathsearch section
- Templatesearch Section

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Aliases Section

An alias converts the name that is initially requested into a different name. It's a great way to universally associate a DataType with a custom named Field Template. It's like a UIHint, except globally defined.

The `<aliases>` section contains a list of `<alias>` tags.

```
<aliases>
  <alias origname="name"
    namesource="DataType|UIHint|Any"
    modes="readonly|edit|insert"
    newname="name"
    enabled="true|false" />
</aliases>
```

The idea is to map origname to newname. There are two properties to determine when the mapping is used: namesource and modes.

This example maps from "Date" to "DatePicker" Field Templates when the DataTypeAttribute or ColumnType determines the column uses a "Date" Field Template. It also requires the DataBound control to be in either Edit or Insert mode.

```
<aliases>
  <alias origname="Date"
    namesource="DataType"
    modes="edit|insert"
    newname="DatePicker" />
</aliases>
```

<alias> Attributes

Defines one alias with the following attributes:

- **origname** – String. Required. The name of the requested Field Template. It can be determined from either the datatype or UIHint, based on the next attribute. Should not include the mode ("_Edit" or "_Insert") or the file extension.
- **namesource** - Enumerated. Optional. Determines if the original name is used based on datatype and UIHint. Its values are:
  - DataType – The name cannot be coming from the UIHint. It must be only from the DataTypeAttribute or ColumnType.
  - UIHint – The name must come from the UIHint. It cannot be based on the DataTypeAttribute or ColumnType
  - Any – The name can be from DataType, ColumnType, or UIHint. This is the default when namesource is not provided.
- **modes** – String. Optional. Determines whether its used for ReadOnly, Edit or Insert modes. The string is a pipe (|) delimited list of these terms: `readonly`, `edit`, `insert`. (Very much like the **Action** property used in a Route declaration.) If not supplied, all modes are supported.
- **newname** – String. Required. Determines the Field Template name that will be used. Should not include the mode ("_Edit" or "_Insert") or the file extension. This name will be processed by the <templatesearch> tags if used.
- **enabled** – Boolean. Optional. Determines if this tag is used. Here to allow quickly turning off the feature (in other words, expect to see enabled=false but not enabled=true). When not declared, it defaults to true.

Note: Anytime the FieldTemplates.config file is changed, be sure to restart your web application as Dynamic Data caches related data.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Pathsearch section

Defines paths to folders that contain Field Templates. It provides a way to order which folders are searched when they contain the same Field Template filenames.

The `<pathsearch>` tag contains a list of `<path>` tags that specify folders to search.

```
<pathsearch>
  <path url="url" />
</pathsearch>
```

The first folder to contain the requested Field Template is used in a first to last `<path>` search order.

Each folder that contains Field Templates should have an entry here although the default DynamicData folder is always known by the pathname of the empty string, so it can be omitted. If it is not defined here, it is searched last. If it is defined here, it's url is usually "".

If the requested Field Template has an alias, the alias has already been processed and the new name is used in searching `<templatesearch>`.

This example searches ~/DynamicData/FieldTemplates, then ~/DynamicData/FieldTemplates/RadControls, then ~/DynamicData/CustomFieldTemplates.

```
<pathsearch>
  <path url="Field Templates" />
  <path url="Field Templates/RadControls" />
  <path url="CustomFieldTemplates" />
</pathsearch>
```

<path> Attributes

- **url** – String. Required. A relative URL defining the location of the folder containing Field Templates.

  If it starts with a ~, it is relative to the web application root. If it starts with a folder name (no lead slash), it is relative to the **DynamicData** folder. This is the recommend usage.

  For example url="Field Templates" really is "~/DynamicData/FieldTemplates/". Trailing slash is not needed.

  Lead slash (/) is not a good idea because it will force the path from the domain root, which is non-existant when using the ASP.NET Development server.

Note: Anytime the FieldTemplates.config file is changed, be sure to restart your web application as Dynamic Data caches related data.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Templatesearch Section

Determines a specific folder containing the requested Field Template. This overrides the `<pathsearch>` rules.

The `<templatesearch>` section contains a list of `<template>` tags.

```
<templates>
  <template name="Field Template name"
    url="name"
    modes="readonly|edit|insert"
    enabled="true|false" />
</templates>
```

Since there can be duplicates of the same template, the first found is used in a first to last search.

If the requested Field Template has an alias, the alias has already been processed and the new name is used in searching `<templatesearch>`.

This example maps the Field Template for "Integer" to one for RadControls, **[web application]/DynamicData/FieldTemplates/RadControls/Integer.ascx**. (Normally, `<pathsearch>` would have used **[web application]/DynamicData/FieldTemplates/Integer.ascx**.)

```
<template name="Integer" url="Field Templates/RadControls" modes="ReadOnly" />
```

<template> Attributes

Maps one Field Template to a URL with these attributes:

- **name** – String. Required. The name of the requested Field Template. Should not include the mode ("_Edit" or "_Insert") or the file extension.
- **url** – String. Required. A relative URL defining the location of the folder containing the desired Field Templates.

  If it starts with a ~, it is relative to the web application root. If it starts with a folder name (no lead slash), it is relative to the **DynamicData** folder. This is the recommend usage.

  For example url="Field Templates" really is "~/DynamicData/FieldTemplates/". Trailing slash is not needed.

  Lead slash (/) is not a good idea because it will force the path from the domain root, which is non-existant when using the ASP.NET Development server.
- **modes** – String. Optional. Determines whether its used for ReadOnly, Edit or Insert modes. The string is a pipe (|) delimited list of these terms: `readonly`, `edit`, `insert`. (Very much like the **Action** property used in a Route declaration.) If not supplied, all modes are supported.
- **enabled** – Boolean. Optional. Determines if this tag is used. Here to allow quickly turning off the feature (in other words, expect to see enabled=false but not enabled=true). When not declared, it defaults to true.

Note: Anytime the FieldTemplates.config file is changed, be sure to restart your web application as Dynamic Data caches related data.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Understanding how Field Templates are selected

You can develop more than one Field Template that supports a data type. For example, a double value is supported by **Decimal.ascx**, **Currency.ascx**, **Percent.ascx**, **Duration.ascx**, and **Measurement.ascx**. By using a DataTypeAttribute on the column's metadata, it loads the more specific Field Template, such as PercentDataTypeAttribute selects **Percent.ascx**. Yet that level of selectability is not enough. Suppose you have two ways to display or edit Percentage. In that case, you create an additional Field Template for the alternative interface and give it a different name. Use the UIHintAttribute, DynamicField, and DynamicControl to specify the name of the alternative Field Template.

The FieldTemplateFactory class, which is assigned to the MetaModel in `Application_Start()`, makes the decisions on which Field Template to use. If it cannot find the right template for a UIHintAttribute or DataTypeAttribute, it has fallback rules to ensure that some kind of template is provided. DES provides its own FieldTemplateFactory, `PeterBlum.DES.DynamicData.DESFieldTemplateFactory`, to allow more customization in this process by supplying the **FieldTemplates.config** file where you specify your rules. See "FieldTemplateFactory: Selecting the correct Field Template file".

Here is the order the DESFieldTemplateFactory uses to determine the right Field Template:

1. Modify the incoming UIHint value coming from either the UIHintAttribute or the **UIHint** property on the DynamicField or DynamicColumn.
   - If you have a UIHint specified, look at the `<aliases>` section of the **FieldTemplates.config** file to see if an alternative UIHint should be used.
   - If the column is a ForeignKey, and there is not UIHint specified, use the Field Template called "ForeignKey".
   - If the column is has a one-to-many relationship, and there is not UIHint specified, use the Field Template called "Children".
2. Look for a match to the UIHint.
3. Look for a match to the DataTypeAttribute. It calls the `DataTypeAttribute.GetDataTypeName()` method.
4. Look for a match to the full name of the column's property Type, such as "System.Int32" for a 32 bit integer. *Note: This rule can be skipped by setting DESFieldTemplateFactory.FallbackTestsFullNameType to false.*
5. Look for a match to the short name of the column's property Type, such as "Int32" for a 32 bit integer.
6. Convert these types to these strings and look for a match:
   - Int32 → "Integer"
   - String → "Text"
7. Convert the type itself to a compatible type and run through the above steps again.
   - float, double → decimal (System.Decimal)
   - Int16, Byte, Long → Int32
   - Char, Int32, decimal, GUID, DateTime, DateTimeOffset, TimeSpan → String

Additional Rules

1. If using Insert mode and there is no template name for *name*_Insert.ascx, look for *name*_Edit.ascx, when **DESFieldTemplateFactory.FallbackInsertToEditMode** is `true`. It defaults to `true`.
2. If using Edit mode and there is no template name for *name*_Edit.ascx, look for *name*.ascx when **DESFieldTemplateFactory.FallbackEditToReadOnlyMode** is `true`. That property defaults to `false` so this rule is not applied by default. *Note: It can fallback from Insert mode all the way to ReadOnly mode if there is only a Field Template for readonly mode.*
3. If you have setup an `<alias>` in the **FieldTemplates.config** file, it will be applied to the name determined by each of the above steps before testing the file system for the presence of the Field Template file. For example, if it attempts to find "Int32" and you have made an `<alias>` that maps "Int32" to "Integer", always expect step 5 to use Integer for any Int32 type columns.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Creating Field Templates programmatically

There are times where you want to create the Field Templates programmatically instead of letting them get loaded from UserControl files.

- You don't want to use the Field Templates folder, making your application more lightweight. The DESFieldTemplateFactory can instantiate the correct FieldTemplateUserControl class for the given column and populate its with a default set of controls. See "Abandoning the Field Templates folder".
- You want to improve performance. UserControl files require loading and parsing. Instead, create the appropriate FieldTemplateUserControl class and populate its Controls collection. See "Abandoning the Field Templates folder".
- You have a special case where a column needs a different FieldTemplateUserControl class or controls collection. You will handle that programmatically, while letting DESFieldTemplateFactory handle the other cases. See "Using the CreateTemplate event handler"'.

The DESFieldTemplateFactory offers the **CreateTemplate** event handler to assist with all of these. It calls a method that can create a FieldTemplateUserControl object. If is does not create one, the default code of the DESFieldTemplateFactory executes.

Abandoning the Field Templates folder

If you want to abandon the Field Templates folder, create the DESMetaModel object passing one parameter, true. It sets up the DESFieldTemplateFactory with the **CreateTemplate** event handler setup to create FieldTemplateUserControls programmatically.

The following shows how to update that code. If using the `RegisterRoutes()` method, the edits go there. Otherwise, they go directly in `Application_Start()`.

[C#]

```
PeterBlum.DES.DynamicData.DESMetaModel model =
  new PeterBlum.DES.DynamicData.DESMetaModel(true);
```

[VB]

```
Dim model As PeterBlum.DES.DynamicData.DESMetaModel = _
  New PeterBlum.DES.DynamicData.DESMMetaModel(True)
```

How this works

DES contains numerous subclasses of `PeterBlum.DES.DynamicData.DESFieldTemplateUserControl` to handle the common cases for datatypes. See "Field Template classes and properties". DES sets up a list of UIHint string values and the associated DESFieldTemplateUserControl subclasses for read-only, edit, and insert mode. When the UIHint is requested, it locates the associated class for the given mode and instantiates it. Finally, it calls the `CreateDefaultControls()` method on that class, which is designed to mimic the UserControl file with its factory configuration.

The UIHint value comes from several sources at runtime:

- UIHintAttribute or a **UIHint** property on the DynamicControl or DynamicField.
- DataTypeAttribute by converting its datatype value to a string. For example, System.ComponentModel.DataAnnotations.DataType.MultilineText becomes "MultilineText".
- Information about the datatype of the MetaColumn. Integer types become "Integer", floating point types become "Decimal", etc.

You can change the FieldTemplateUserControl classes used by the known UIHints. You can also introduce your own UIHints and map them to FieldTemplateUserControls. See also "Adjusting the list that maps UIHint to FieldTemplateUserControl type".

If the existing FieldTemplateUserControl classes do not supply the desired look, see "Changing the controls output by a FieldTemplateUserControl".

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Adjusting the list that maps UIHint to FieldTemplateUserControl type

The default list knows only about the standard types. It does not know about any value you assign to as a UIHint, such as "Calendar" used instead of "Date" to get a calendar editor for date values. You must add support for your own UIHints.

In addition, you may want to replace a known UIHint value with different classes. Perhaps you have subclassed the FieldTemplateUserControl to implement the `CreateDefaultControls()` method your own way and want to use that.

In both cases, call the `DESMetaModel.FieldTemplateFactory.MapIdentifierToFieldTemplateTypes()` method within the `RegisterRoutes()` or `Application_Start()` method. Here are the definitions for that method:

[C#]

```
void MapIdentifierToFieldTemplateTypes(string pIdentifier,
   Type pReadOnlyFieldTemplateType, Type pEditFieldTemplateType,
   Type pInsertFieldTemplateType)
```

```
void MapIdentifierToFieldTemplateTypes(string pIdentifier,
   Type pReadOnlyFieldTemplateType, Type pEditFieldTemplateType)
```

[VB]

```
Sub MapIdentifierToFieldTemplateTypes(ByVal pIdentifier As String, _
   ByVal pReadOnlyFieldTemplateType As Type, ByVal pEditFieldTemplateType As Type,
   ByVal pInsertFieldTemplateType As Type)
```

```
Sub MapIdentifierToFieldTemplateTypes(ByVal pIdentifier As String,
   ByVal pReadOnlyFieldTemplateType As Type, ByVal pEditFieldTemplateType As Type)
```

Parameters

pIdentifier

The UIHint or other identifier that maps to the classes for FieldTemplateUserControls. See "How this works".

pReadOnlyFieldTemplateType

The type for the FieldTemplateUserControl used in ReadOnly mode.

pEditFieldTemplateType

The type for the FieldTemplateUserControl used in Edit mode.

pInsertFieldTemplateType

The type for the FieldTemplateUserControl used in Insert mode. When not specified, it uses the value from *pEditFieldTemplateType*.

Example 1: Adding support for a new UIHint

This example declares "Calendar" that maps to DateStaticFTUC for readonly mode and CalendarEditFTUC for edit and insert modes.

[C#]

```
using DES= PeterBlum.DES.DynamicData;
...
DES.DESMetaModel model = new DES.DESMetaModel(true);
model.FieldTemplateFactory.MapIdentifierToFieldTemplateTypes("Calendar",
   typeof(DES.DateStaticFTUC), typeof(DES.CalendarEditFTUC));
```

[VB]

```
Imports DES= PeterBlum.DES.DynamicData
...
Dim model As DES.DESMetaModel =  New DES.DESMMetaModel(True)
model.FieldTemplateFactory.MapIdentifierToFieldTemplateTypes("Calendar",
   GetType(DES.DateStaticFTUC), GetType(DES.CalendarEditFTUC))
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Example 2: Changing the CreateDefaultControls method

This example modifies how controls are created for the Integer datatype in Edit mode by subclassing `PeterBlum.DES.DynamicData.IntegerEditFTUC` and overrriding `CreateDefaultControls()`.

You basically do the same work you would in the Page_Init() method of the UserControl file, creating controls and registering them with the DESFieldTemplateUserControl class.

[C#]

```
public class IntegerEditFTUCWithCSS : PeterBlum.DES.DynamicData.IntegerEditFTUC
{
  public override IFieldTemplate CreateDefaultControls()
  {
    IntegerTextBox vIntegerTextBox = new IntegerTextBox();
    vIntegerTextBox.ID = "Value";
    vIntegerTextBox.Columns = 10;
    vIntegerTextBox.CssClass = "IntegerCss";
    Controls.Add(vIntegerTextBox);
    SetUpEditableDataControl(vIntegerTextBox);

    CreateColumnValidatorManager();
  }
}
```

In Application_Start() or RegisterRoutes:

```
using DES= PeterBlum.DES.DynamicData;
...
DES.DESMetaModel model = new DES.DESMetaModel(true);
model.FieldTemplateFactory.MapIdentifierToFieldTemplateTypes("Integer",
  typeof(DES.IntegerStaticFTUC), typeof(IntegerEditFTUCWithCSS));
```

[VB]

```
Public Class IntegerEditFTUCWithCSS _
  Inherits PeterBlum.DES.DynamicData.IntegerEditFTUC
  Public Overrides Function CreateDefaultControls() As IFieldTemplate

    Dim vIntegerTextBox As IntegerTextBox = New IntegerTextBox()
    vIntegerTextBox.ID = "Value"
    vIntegerTextBox.Columns = 10
    vIntegerTextBox.CssClass = "IntegerCss"
    Controls.Add(vIntegerTextBox)
    SetUpEditableDataControl(vIntegerTextBox)

    CreateColumnValidatorManager()
  End Function
End Class
```

In Application_Start() or RegisterRoutes:

```
Imports DES= PeterBlum.DES.DynamicData
...
Dim model As DES.DESMetaModel =  New DES.DESMMetaModel(True)
model.FieldTemplateFactory.MapIdentifierToFieldTemplateTypes("Integer", _
  GetType(DES.IntegerStaticFTUC), GetType(IntegerEditFTUCWithCSS));
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Changing the controls output by a FieldTemplateUserControl

If the existing FieldTemplateUserControl classes do not supply the desired look, do the following:

1. Subclass from the best fit FieldTemplateUserControl. See "Field Template classes and properties".
2. Override the `CreateDefaultControls()` method.
3. Create the desired controls within that method.
   - Always assign the ID property to controls that you create. For controls that show data, the suggested ID is "Value". (If you happen to have multiple data controls, do not assign more than one to ID "Value".)
   - When adding the control that shows data in read-only mode, call `SetUpStaticDataControl(control)`.
   - When adding the control that shows data in edit or insert modes, call `SetUpEditableDataControl(control)`.
   - When adding the ColumnValidatorManager control, call `CreateColumnValidatorManager()`. The control is added to the end of the Field Template's Controls collection. It returns an instance of the ColumnValidatorManager in case you want to edit the control's properties.
   - When adding a Label control that shows data, call CreateLabelForStaticData().It creates a LocalizableLabel control and adds it to the end of the Field Template's Controls collection. It returns an instance of the Label in case you want to edit the control's properties. It calls `SetUpStaticDataControl()` for you.
   - The BaseTextBoxFTUC class offers the method `CreateTextBox(filtered)`. It creates a textbox control and adds it to the end of the Field Template's Controls collection. It returns an instance of the TextBox in case you want to edit the control's properties. It calls `SetUpEditableDataControl()` for you.

     The parameter, filtered, is a boolean. When `true`, it creates a FilteredTextBox. When `false`, it creates a DES.TextBox. BaseTextBoxFTUC is the ancestor to TextEditFTUC, MultilineTextEditFTUC, PasswordEditFTUC, and EmailAddressEditFTUC.
   - The BaseTextBoxFTUC class also offers the method `CreateTextCounter(textbox)`. It creates a TextCounter control and adds it to the end of the Field Template's Controls collection. It returns an instance of the TextCounter in case you want to edit the control's properties. It calls `SetUpTextCounter()` for you.

     The parameter, textbox, is the TextBox control that it will monitor.
4. Register your new class in the `Application_Start()` or `RegisterRoutes()` method using the `DESMetaModel.FieldTemplateFactory.MapIdentifierToFieldTemplateTypes()` method. See "Adjusting the list that maps UIHint to FieldTemplateUserControl type". For an example, see the previous page.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Using the CreateTemplate event handler

The CreateTemplate event handler is located on the DESFieldTemplateFactory object. It must be assigned in the `RegisterRoutes()` or `Application_Start()` method like this:

[C#]

```
using DES= PeterBlum.DES.DynamicData;
...
DES.DESMetaModel model = new DES.DESMetaModel();
model.FieldTemplateFactory.CreateTemplate =
   new DES.CreateTemplateEventHandler(yourmethod);
```

[VB]

```
Imports DES= PeterBlum.DES.DynamicData
...
Dim model As DES.DESMetaModel =  New DES.DESMMetaModel()
model.FieldTemplateFactory.CreateTemplate =
   New DES.CreateTemplateEventHandler(AddressOf yourmethod)
```

Your method must follow the CreateTemplateEventHandler delegate:

[C#]

```
public delegate void CreateTemplateEventHandler(CreateFieldTemplateArgs pArgs);
```

[VB]

```
Public Delegate Sub CreateTemplateEventHandler( _
   ByVal pArgs As CreateFieldTemplateArgs)
```

Your method's task is to evaluate the **Mode**, **Column**, and **UIHint** properties of pArgs to determine what is requested. If your method can handle it, it creates the FieldTemplateUserControl object and assigns that object to **pArgs.FieldTemplateCreated**.

Example

When the column is "Product.UnitPrice" in Edit or Insert mode, create a CurrencyEditFTUC whose CurrencyTextBox uses the style sheet class "UnitPrice".

[C#]

```
using DES= PeterBlum.DES.DynamicData;
...
DES.DESMetaModel model = new DES.DESMetaModel();
model.FieldTemplateFactory.CreateTemplate =
   new DES.CreateTemplateEventHandler(MyCreateTemplates);

protected void MyCreateTemplates(DES.CreateFieldTemplateArgs pArgs)
{
   if ((pArgs.Column.Name == "UnitPrice") &&
       (pArgs.Column.Table.Name == "Product") &&
       (pArgs.Mode != DataBoundControlMode.ReadOnly))
   {
      DES.CurrencyEditFTUC vFTUC = new DES.CurrencyEditFTUC();
      pArgs.FieldTemplateCreated = vFTUC;
```

EXAMPLE CONTINUES ON THE NEXT PAGE

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

```
// populate the controls collection
     CurrencyTextBox vCurrencyTextBox = new CurrencyTextBox();
     vCurrencyTextBox.ID = "Value";
     vCurrencyTextBox.CssClass = "UnitPrice";
     vFTUC.Controls.Add(vCurrencyTextBox);
     vFTUC.SetUpEditableDataControl(vCurrencyTextBox);

     vFTUC.CreateColumnValidatorManager();
   }
}
```

[VB]

```
Imports DES= PeterBlum.DES.DynamicData
...
Dim model As DES.DESMetaModel =  New DES.DESMMetaModel()
model.FieldTemplateFactory.CreateTemplate =
   New DES.CreateTemplateEventHandler(AddressOf MyCreateTemplates)

Protected Sub MyCreateTemplates(ByVal pArgs DES.CreateFieldTemplateArgs)
   If (pArgs.Column.Name = "UnitPrice") And _
      (pArgs.Column.Table.Name = "Product") And _
      (pArgs.Mode <> DataBoundControlMode.ReadOnly) Then

     ByVal vFTUC As DES.CurrencyEditFTUC = New DES.CurrencyEditFTUC()
     pArgs.FieldTemplateCreated = vFTUC

' populate the controls collection
     ByVal vCurrencyTextBox As CurrencyTextBox = New CurrencyTextBox()
     vCurrencyTextBox.ID = "Value"
     vCurrencyTextBox.CssClass = "UnitPrice"
     vFTUC.Controls.Add(vCurrencyTextBox)
     vFTUC.SetUpEditableDataControl(vCurrencyTextBox)

     vFTUC.CreateColumnValidatorManager()
   End If
End Sub
```

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Assistance on Related Technologies

ASP.NET Dynamic Data uses several technologies extensively: ASP.NET Routing, LINQ to SQL, Entity Framework. The real concept of ASP.NET Dynamic Data itself is the DynamicField and DynamicControl, which deliver a user interface (Field Template) based on metadata describing a column. This User's Guide focuses on that and PeterBlum.com provides technical support for it. As for the other technologies, this section should help you find the right resources. PeterBlum.com technical support does not provide support for them.

Please submit links to articles on DynamicData, LINQ to SQL, Entity Framework and other related technologies to plblum@peterblum.com for inclusion in this section.

Click on any of these topics to jump to them:

- ASP.NET Dynamic Data
- LINQ to SQL
- ASP.NET Routing (System.Web.Routing namespace)

ASP.NET Dynamic Data

WARNING: There is a lot of out-of-date documentation available! Dynamic Data's design has changed dramatically since its early releases (December 2007) and the official release (July 2008). Please be sure you are using documentation that reflects the official ASP.NET 3.5 SP 1 release. This warning even applies to what Microsoft has published (including videos) and to blogs.

Home page: http://www.asp.net/dynamicdata

Videos: http://www.asp.net/learn/3.5-SP1/

Forums

The single most important place to ask general questions about ASP.NET Dynamic Data is the Dynamic Data forum at forums.asp.net.

Do not use it for DES Dynamic Data questions. Use PeterBlum.com tech support for that.

Relevant Blogs

Scott Hunter, Microsoft, Project Manager for ASP.NET Dynamic Data.

David Ebbo, Microsoft, Lead Architect for ASP.NET Dynamic Data.

Marcin Dobosz, Microsoft, Engineer for ASP.NET Dynamic Data.

Stephen Naughton, "C# Bits"

Jonathan Carter, "Lost In Tangent"

Articles

Note: PeterBlum.com cannot provide technical support on these areas. This section is here to help you find the available information. It was published before Dynamic Data was RTMed, so use web searches to find more recent information.

Limiting Access to your PageTemplates

Mike Ormond, Microsoft, "ASP.NET Routing and Authorization"

Stephen Naughton, "A DynamicData Attribute Based Permission Solution using User Roles"

Other

Matt Berseth, "Dynamic Data and Custom Metadata Providers"

Matt Berseth, "A Dynamic Menu for your Dynamic Data"

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

LINQ to SQL

Note: PeterBlum.com cannot provide technical support on these areas. This section is here to help you find the available information.

Home page: http://msdn.microsoft.com/en-us/netframework/aa904594.aspx

Forums

The LINQ Project General forum at forums.microsoft.com.

Blogs

See "LINQ Bloggers" on http://msdn.microsoft.com/en-us/netframework/aa904594.aspx

Articles

MSDN, "Using LINQ with ASP.NET"

Scott Guthrie, Microsoft, VP. "LINQ to SQL"

- Part 1: Introduction to LINQ to SQL
- Part 2: Defining our Data Model Classes
- Part 3: Querying our Database
- Part 4: Updating our Database
- Part 5: Binding UI using the ASP:LinqDataSource Control

Don Box, Anders Hejlsberg, "LINQ: .NET Language-Integrated Query".

Scott Guthrie, Microsoft, VP. "LINQ to SQL Debug Visualizer"

Bipin Joshi, BinaryIntellect Consulting, "Using LINQ in ASP.NET" Part1 Part2 Part3 Part4

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

ASP.NET Routing (System.Web.Routing namespace)

Note: PeterBlum.com cannot provide technical support on these areas. This section is here to help you find the available information..

Articles

Phil Haack, Microsoft, Senior Program Manager, "ASP.NET Routing Debugger".

Phil Haack, Microsoft, Senior Program Manager, "Using Routing with WebForms" and "Updated Routing with WebForms".

Mike Ormond, Microsoft, "Using ASP.NET Routing Independent of MVC".

Mike Ormond, Microsoft, "ASP.NET Routing and Authorization"

Joe Stagner, Microsoft, "How to Enable Table Specific Routing in Dynamic Data Applications" (video)

David Ebbo, Microsoft, "Using Dynamic Data with Multiple Databases"

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved

Troubleshooting

Here are some issues that you may run into. Remember that technical support is available from support@PeterBlum.com. We encourage you to use this knowledge base first.

This guide contains problems specific to the **DES Dynamic Data**. Please see the "Troubleshooting" section of the **General Features Guide** for an extensive list of other topics including "Handling JavaScript Errors" and "Common Error Messages".

None specific to this module. Please use the General Features Guide.

Copyright © 2002-2010 Peter L. Blum. All Rights Reserved