Elevate Web Builder 2 Manual

Table Of Contents

Chapter 1 - Getting Started **1**

1.1 System Requirements 1

1.2 General Architecture 2

1.3 Application Structure 4

1.4 Compiling Applications 6

1.5 Component Library 8

1.6 Visual Applications 10

1.7 Control Interfaces 14

1.8 Icon Library 18

1.9 Accessing Help 21

1.10 Example Applications 26

Chapter 2 - Using the IDE **37**

2.1 Introduction 37

2.2 Creating a New Project 39

2.3 Adding to an Existing Project 42

2.4 Modifying Project Options 45

2.5 Compiling a Project 55

2.6 Deploying a Project 57

2.7 Running a Project 58

2.8 Saving a Project 60

2.9 Viewing Project Forms and Databases 61

2.10 Viewing Project Units 62

2.11 Using the Object Inspector 63

2.12 Using the Form and Database Designers 66

2.13 Using the Code Editor 70

2.14 Using the Project Manager 78

2.15 Using the Database Manager 83

2.16 Viewing Messages 91

2.17 Modifying Environment Options 93

2.18 Creating a New Component 109

2.19 Adding a Component to the Component Library 110

2.20 Removing a Component from the Component Library 112

2.21 Rebuilding the Component Library 114

2.22 Creating a New Control Interface 115

2.23 Modifying a Control Interface 116

2.24 Using the Control Interface Editor 117

2.25 Opening the Icon Library 123

Chapter 3 - Using Visual Controls **125**

3.1 Standard Controls 125

3.2 Creating and Showing Forms 128

3.3 Showing Message Dialogs 131

3.4 Showing Progress Dialogs 133

3.5 Using HTML Forms 134

3.6 Layout Management 136

Chapter 4 - Using Server Requests **149**

4.1 Server Request Architecture 149

4.2 Executing a Server Request 153

Chapter 5 - Using Local Storage **157**

5.1 Introduction 157

5.2 Saving Data To Local Storage 158

5.3 Loading Data from Local Storage 159

5.4 Detecting Local Storage Changes 160

Chapter 6 - Using Databases **163**

6.1 Database Architecture 163

6.2 Creating and Using Databases 166

6.3 Creating and Loading DataSets 168

6.4 Navigating DataSets 173

6.5 Searching and Sorting DataSets 176

6.6 Updating DataSets 178

6.7 Transactions 182

6.8 Responding to DataSet Changes 186

6.9 Binding Controls to DataSets 189

6.10 Calculated Columns 191

6.11 API Reference 192
6.12 JSON Reference 194
Chapter 7 - Using the Web Server **201**
7.1 Starting the Web Server 201
7.2 Configuring the Web Server 203
7.3 Multiple Web Server Instances 223
7.4 Web Server Request Handling 225
7.5 Creating Web Server Modules 228
Chapter 8 - Language Reference **231**
8.1 Introduction 231
8.2 Defines 235
8.3 Types 237
8.4 Operators 240
8.5 Statements 242
8.6 Units 249
8.7 Constant Declarations 254
8.8 Type Declarations 255
8.9 Variable Declarations 257
8.10 Function and Procedure Declarations 258
8.11 Function and Procedure Implementations 259
8.12 Enumerations 261
8.13 Arrays 262
8.14 Classes 265
8.15 Variables (In Classes) 268
8.16 Methods 270
8.17 Properties 277
8.18 Events 282
8.19 Scope 286
8.20 Casting Types 290
8.21 Exception Handling 292
8.22 External Interfaces 296
8.23 Debugging 299
8.24 Asynchronous Calls 301
Chapter 9 - Function and Procedure Reference **303**

9.1 Abs 303

9.2 ArcCos 304

9.3 ArcSin 305

9.4 ArcTan 306

9.5 ArcTan2 307

9.6 Assigned 308

9.7 BoolToStr 309

9.8 Ceil 310

9.9 Chr 311

9.10 CompareStr 312

9.11 CompareText 313

9.12 Copy 314

9.13 Cos 315

9.14 CreateActiveXObject 316

9.15 CreateObject 317

9.16 Date 318

9.17 DateTimeToStr 319

9.18 DateTimeToISOStr 320

9.19 DateToStr 321

9.20 DayOf 322

9.21 Dec 323

9.22 Degrees 324

9.23 Delete 325

9.24 DoubleToStr 326

9.25 EncodeDate 327

9.26 EncodeDateTime 328

9.27 EncodeTime 329

9.28 Exp 330

9.29 FloatToStr 331

9.30 Floor 332

9.31 HideProgress 333

9.32 HourOf 334

9.33 Inc 335

9.34 Insert 336

9.35 IntToHex 337
9.36 IntToStr 338
9.37 ISOStrToDateTime 339
9.38 Join 340
9.39 Length 341
9.40 Ln 342
9.41 LocaleCompareStr 343
9.42 LocaleCompareText 344
9.43 LocaleLowerCase 345
9.44 LocaleSameStr 346
9.45 LocaleSameText 347
9.46 LocaleUpperCase 348
9.47 LowerCase 349
9.48 Max 350
9.49 MessageDlg 351
9.50 Min 353
9.51 MinuteOf 354
9.52 MonthOf 355
9.53 MSecondOf 356
9.54 Now 357
9.55 Ord 358
9.56 Pad 359
9.57 Pi 360
9.58 ParseXML 361
9.59 Pos 362
9.60 Power 363
9.61 Radians 364
9.62 Random 365
9.63 Round 366
9.64 QuotedStr 367
9.65 SameStr 368
9.66 SameText 369
9.67 SecondOf 370
9.68 SerializeXML 371

9.69 SetLength 372

9.70 ShowMessage 373

9.71 ShowProgress 374

9.72 Sin 375

9.73 Split 376

9.74 Sqrt 377

9.75 StrReplace 378

9.76 StrToBool 379

9.77 StrToDate 380

9.78 StrToDateTime 381

9.79 StrToDouble 382

9.80 StrToFloat 383

9.81 StrToInt 384

9.82 StrToTime 385

9.83 Tan 386

9.84 Time 387

9.85 TimeToStr 388

9.86 TimeZoneOffset 389

9.87 Trim 390

9.88 Trunc 391

9.89 UpperCase 392

9.90 WeekDayOf 393

9.91 YearOf 394

Chapter 10 - Component Reference 395

10.1 TAbstractList Component 395

10.2 TAddress Component 399

10.3 TAlertLabel Component 413

10.4 TAlertLabelControl Component 448

10.5 TAnimatedIcon Component 450

10.6 TAnimation Component 475

10.7 TAnimations Component 482

10.8 TApplication Component 490

10.9 TAudio Component 510

10.10 TAudioElement Component 572

10.11 TAutoSize Component 573
10.12 TBackground Component 577
10.13 TBackgroundImage Component 583
10.14 TBalloonLabel Component 596
10.15 TBalloonLabelControl Component 628
10.16 TBasicPanel Component 629
10.17 TBasicPanelControl Component 663
10.18 TBindableColumnControl Component 664
10.19 TBindableControl Component 665
10.20 TBlobValue Component 666
10.21 TBodyElement Component 667
10.22 TBooleanValue Component 668
10.23 TBorder Component 669
10.24 TBorderSide Component 675
10.25 TBoundingAttribute Component 682
10.26 TBrowser Component 688
10.27 TButton Component 714
10.28 TButtonComboBox Component 746
10.29 TButtonComboControl Component 793
10.30 TButtonControl Component 794
10.31 TButtonInputControl Component 795
10.32 TCalendar Component 796
10.33 TCalendarControl Component 834
10.34 TCanvasElement Component 839
10.35 TCanvasGradient Component 888
10.36 TCanvasPattern Component 899
10.37 TCanvasPoint Component 903
10.38 TCanvasRect Component 911
10.39 TCaptionBarControl Component 931
10.40 TCheckBox Component 932
10.41 TCollection Component 969
10.42 TCollectionItem Component 985
10.43 TComponent Component 992
10.44 TConstraint Component 1004

10.45 TConstraints Component 1008
10.46 TContentLayout Component 1012
10.47 TControl Component 1019
10.48 TCookies Component 1071
10.49 TCorner Component 1080
10.50 TCorners Component 1084
10.51 TDatabase Component 1090
10.52 TDataColumn Component 1121
10.53 TDataColumns Component 1143
10.54 TDataRow Component 1145
10.55 TDataSet Component 1146
10.56 TDataSetController Component 1222
10.57 TDataSetToolBar Component 1228
10.58 TDataSetToolBarButton Component 1245
10.59 TDataSetToolBarButtons Component 1246
10.60 TDataValue Component 1257
10.61 TDataValues Component 1272
10.62 TDateEditComboBox Component 1282
10.63 TDateTimeValue Component 1330
10.64 TDateValue Component 1331
10.65 TDialog Component 1332
10.66 TDialogButton Component 1366
10.67 TDialogCaptionBar Component 1370
10.68 TDialogClient Component 1380
10.69 TDialogControl Component 1384
10.70 TDialogEditComboBox Component 1385
10.71 TDivElement Component 1425
10.72 TDropDownButtonControl Component 1426
10.73 TDropDownEditControl Component 1429
10.74 TEdit Component 1432
10.75 TEditComboBox Component 1472
10.76 TEditComboControl Component 1522
10.77 TEditControl Component 1523
10.78 TElement Component 1528

10.79 TElementAttribute Component 1630

10.80 TElementProperties Component 1634

10.81 TElements Component 1662

10.82 TFileComboBox Component 1670

10.83 TFileInputElement Component 1704

10.84 TFill Component 1707

10.85 TFloatValue Component 1712

10.86 TFont Component 1713

10.87 TFontIcon Component 1721

10.88 TFontStyle Component 1725

10.89 TForm Component 1731

10.90 TFormat Component 1768

10.91 TFormatSettings Component 1773

10.92 TFormControl Component 1794

10.93 TFormElement Component 1800

10.94 TFrameElement Component 1807

10.95 TGradient Component 1814

10.96 TGradientColorStop Component 1822

10.97 TGradientColorStops Component 1826

10.98 TGrid Component 1833

10.99 TGridCell Component 1882

10.100 TGridColumn Component 1887

10.101 TGridControl Component 1929

10.102 TGridHeader Component 1983

10.103 TGridRow Component 1990

10.104 TGridRows Component 1996

10.105 TGroupPanel Component 2002

10.106 TGroupPanelControl Component 2035

10.107 THeaderPanel Component 2036

10.108 THeaderPanelControl Component 2068

10.109 THiddenInputElement Component 2069

10.110 THTMLForm Component 2070

10.111 THTMLFormControl Component 2101

10.112 THTMLLabel Component 2104

10.113 THTMLLabelControl Component 2135

10.114 TIcon Component 2136

10.115 TIconAnimation Component 2158

10.116 TIconButton Component 2161

10.117 TIconControl Component 2186

10.118 TIconLibrary Component 2187

10.119 TIconProperties Component 2190

10.120 TImage Component 2199

10.121 TImageElement Component 2237

10.122 TInputControl Component 2240

10.123 TInputElement Component 2243

10.124 TInsetShadow Component 2255

10.125 TIntegerValue Component 2256

10.126 TInterface Component 2257

10.127 TInterfaceController Component 2261

10.128 TInterfaceManager Component 2264

10.129 TInterfaces Component 2298

10.130 TInterfaceState Component 2305

10.131 TInterfaceStates Component 2308

10.132 TLabel Component 2317

10.133 TLabelControl Component 2351

10.134 TLayout Component 2352

10.135 TLink Component 2359

10.136 TLinkControl Component 2394

10.137 TLinkElement Component 2395

10.138 TListBox Component 2399

10.139 TListControl Component 2449

10.140 TLocation Component 2453

10.141 TLocationServices Component 2465

10.142 TMap Component 2477

10.143 TMapControl Component 2497

10.144 TMapLocation Component 2498

10.145 TMapLocations Component 2507

10.146 TMapOption Component 2509

10.147 TMapOptions Component 2511
10.148 TMapTypeControlMapTypes Component 2536
10.149 TMapTypeControlOptions Component 2541
10.150 TMargins Component 2545
10.151 TMediaControl Component 2546
10.152 TMediaElement Component 2550
10.153 TMenu Component 2571
10.154 TMenuBar Component 2593
10.155 TMenuBarItem Component 2614
10.156 TMenuBarSeparatorItem Component 2637
10.157 TMenuControl Component 2640
10.158 TMenuItem Component 2651
10.159 TMenuItemControl Component 2674
10.160 TMenuSeparatorItem Component 2679
10.161 TMessageDialog Component 2682
10.162 TModalOverlay Component 2720
10.163 TMultiLineEdit Component 2723
10.164 TMultiLineEditControl Component 2769
10.165 TObjectElement Component 2770
10.166 TObjectList Component 2773
10.167 TOutsetShadow Component 2799
10.168 TOverviewMapControlOptions Component 2800
10.169 TPadding Component 2802
10.170 TPage Component 2803
10.171 TPagePanel Component 2834
10.172 TPagePanelControl Component 2853
10.173 TPaint Component 2867
10.174 TPanControlOptions Component 2888
10.175 TPanel Component 2890
10.176 TPanelCaptionBar Component 2935
10.177 TPanelClient Component 2946
10.178 TPanelControl Component 2950
10.179 TParser Component 2952
10.180 TPasswordEdit Component 2980

10.181 TPasswordInputElement Component 3017
10.182 TPersistent Component 3018
10.183 TPersistentStorage Component 3019
10.184 TPlugin Component 3028
10.185 TPoint Component 3052
10.186 TProgressBar Component 3060
10.187 TProgressBarIndicator Component 3087
10.188 TProgressDialog Component 3091
10.189 TRadioButton Component 3119
10.190 TReader Component 3155
10.191 TRect Component 3186
10.192 TRepeatControl Component 3207
10.193 TRotateControlOptions Component 3208
10.194 TScript Component 3210
10.195 TScrollableControl Component 3214
10.196 TScrollPanel Component 3215
10.197 TScrollPanelClient Component 3250
10.198 TScrollPanelControl Component 3254
10.199 TServerRequest Component 3255
10.200 TServerRequestQueue Component 3280
10.201 TSet Component 3288
10.202 TShadow Component 3305
10.203 TSizeGrip Component 3313
10.204 TSizeGripControl Component 3332
10.205 TSizer Component 3333
10.206 TSizerControl Component 3358
10.207 TSlideShow Component 3359
10.208 TSlideShowControl Component 3392
10.209 TStateButtonControl Component 3396
10.210 TStreetViewControlOptions Component 3397
10.211 TStringBuilder Component 3399
10.212 TStringList Component 3408
10.213 TStrings Component 3414
10.214 TStringValue Component 3432

10.215 TSurface Component 3433
10.216 TTab Component 3446
10.217 TTextAreaElement Component 3452
10.218 TTextInputElement Component 3453
10.219 TTimer Component 3455
10.220 TTimeValue Component 3459
10.221 TToolBar Component 3460
10.222 TToolBarButton Component 3476
10.223 TToolBarControl Component 3493
10.224 TVideo Component 3502
10.225 TVideoElement Component 3567
10.226 TViewport Component 3571
10.227 TWebControl Component 3582
10.228 TWebElement Component 3584
10.229 TWriter Component 3591
10.230 TZoomControlOptions Component 3622
Chapter 11 - Type Reference **3625**
11.1 TAlertOrientation Type 3625
11.2 TAnimatedIconDirection Type 3626
11.3 TAnimationCompleteEvent Type 3627
11.4 TAnimationStyle Type 3628
11.5 TAuthenticationMethod Type 3631
11.6 TAutoCompleteType Type 3632
11.7 TBackgroundImageAnimateDirection Type 3633
11.8 TBackgroundImagePositionType Type 3634
11.9 TBackgroundImageRepeatStyle Type 3635
11.10 TBackgroundImageSizeType Type 3636
11.11 TBackgroundOrientationType Type 3637
11.12 TBalloonOrientation Type 3638
11.13 TBooleanArray Type 3639
11.14 TBorderStyle Type 3640
11.15 TCalendarView Type 3641
11.16 TCanPlayMedia Type 3642
11.17 TCanvasPoints Type 3643

11.18 TCaption Type 3644
11.19 TCharArray Type 3645
11.20 TClass Type 3646
11.21 TClickEvent Type 3647
11.22 TCloseQueryEvent Type 3648
11.23 TCollectionItemClass Type 3649
11.24 TCollectionItemName Type 3650
11.25 TColor Type 3651
11.26 TComponentClass Type 3652
11.27 TComponentName Type 3653
11.28 TCompositeOperation Type 3654
11.29 TContent Type 3656
11.30 TContentAlignment Type 3657
11.31 TContentDirection Type 3658
11.32 TContentPosition Type 3659
11.33 TContentSize Type 3660
11.34 TControlClass Type 3661
11.35 TCursor Type 3662
11.36 TDatabaseClass Type 3664
11.37 TDatabaseErrorEvent Type 3665
11.38 TDatabaseEvent Type 3666
11.39 TDataColumnTextEvent Type 3667
11.40 TDataRowEvent Type 3668
11.41 TDataSetErrorEvent Type 3669
11.42 TDataSetEvent Type 3670
11.43 TDataSetState Type 3671
11.44 TDataType Type 3672
11.45 TDateTimeFormat Type 3673
11.46 TDrawStyle Type 3674
11.47 TDropDownPosition Type 3675
11.48 TElementClass Type 3676
11.49 TErrorEvent Type 3677
11.50 TFillType Type 3678
11.51 TFormControlClass Type 3679

11.52 TGenericFontFamily Type 3680
11.53 TGradientType Type 3681
11.54 TGridColumnCellEvent Type 3682
11.55 TGridColumnCompareEvent Type 3683
11.56 TGridColumnControlType Type 3684
11.57 TGridHeaderClickEvent Type 3685
11.58 THTMLFormEncoding Type 3686
11.59 THTMLFormMethod Type 3687
11.60 TIntegerArray Type 3688
11.61 TInterfaceAnimationEvent Type 3689
11.62 TInterfaceControllerClass Type 3690
11.63 TInterfaceErrorEvent Type 3691
11.64 TInterfaceIdleEvent Type 3692
11.65 TInterfaceTimeoutEvent Type 3693
11.66 TInterfaceTimerEvent Type 3694
11.67 TInterfaceViewportResizeEvent Type 3695
11.68 TInterfaceViewportScrollEvent Type 3696
11.69 TKeyDownEvent Type 3697
11.70 TKeyPressEvent Type 3698
11.71 TKeyUpEvent Type 3699
11.72 TLayoutConsumption Type 3700
11.73 TLayoutOverflow Type 3701
11.74 TLayoutPosition Type 3702
11.75 TLayoutStretch Type 3704
11.76 TLineCapStyle Type 3705
11.77 TLineJoinStyle Type 3706
11.78 TLocationError Type 3707
11.79 TMapControlPosition Type 3708
11.80 TMapTilt Type 3709
11.81 TMapType Type 3710
11.82 TMapTypeControlStyle Type 3711
11.83 TMediaNetworkState Type 3712
11.84 TMediaPreload Type 3713
11.85 TMediaReadyState Type 3714

11.86 TModalResult Type 3715
11.87 TMouseDownEvent Type 3716
11.88 TMouseMoveEvent Type 3717
11.89 TMouseUpEvent Type 3718
11.90 TMouseWheelEvent Type 3719
11.91 TMsgDlgBtn Type 3720
11.92 TMsgDlgBtns Type 3721
11.93 TMsgDlgResultEvent Type 3722
11.94 TMsgDlgType Type 3723
11.95 TNotifyEvent Type 3724
11.96 TObjectsArray Type 3725
11.97 TOverflowType Type 3726
11.98 TPageChangeEvent Type 3727
11.99 TPatternRepeatStyle Type 3728
11.100 TRequestMethod Type 3729
11.101 TScrollBars Type 3730
11.102 TScrollSupport Type 3731
11.103 TSelectionState Type 3732
11.104 TServerRequestEvent Type 3733
11.105 TServerRequestProgressEvent Type 3734
11.106 TServerRequestURL Type 3735
11.107 TSizerOrientation Type 3736
11.108 TSlideEvent Type 3737
11.109 TSortDirection Type 3738
11.110 TStorageChangeEvent Type 3739
11.111 TStringsArray Type 3740
11.112 TTextAlignment Type 3741
11.113 TTextBaseLine Type 3742
11.114 TTextInputType Type 3743
11.115 TTouchEvent Type 3744
11.116 TTouchScrollEvent Type 3745
11.117 TWebElementEvent Type 3746
11.118 TZoomControlStyle Type 3747

Chapter 1
Getting Started

1.1 System Requirements

IDE Requirements

The Elevate Web Builder IDE requires the following:

- Windows Vista or higher
- 1024x768 or higher display resolution (widescreen display highly recommended)
- 32-bit color display adapter
- 1GB of installed memory
- 128MB of disk space
- Internet Explorer 9 or higher

Runtime Browser Compatibility

Applications created with Elevate Web Builder work with the following browsers:

Browser	Minimum Version Required
Internet Explorer (Microsoft)	9
FireFox (Mozilla)	5
Chrome (Google)	10
Safari (Apple)	5
Opera (Opera Software)	11

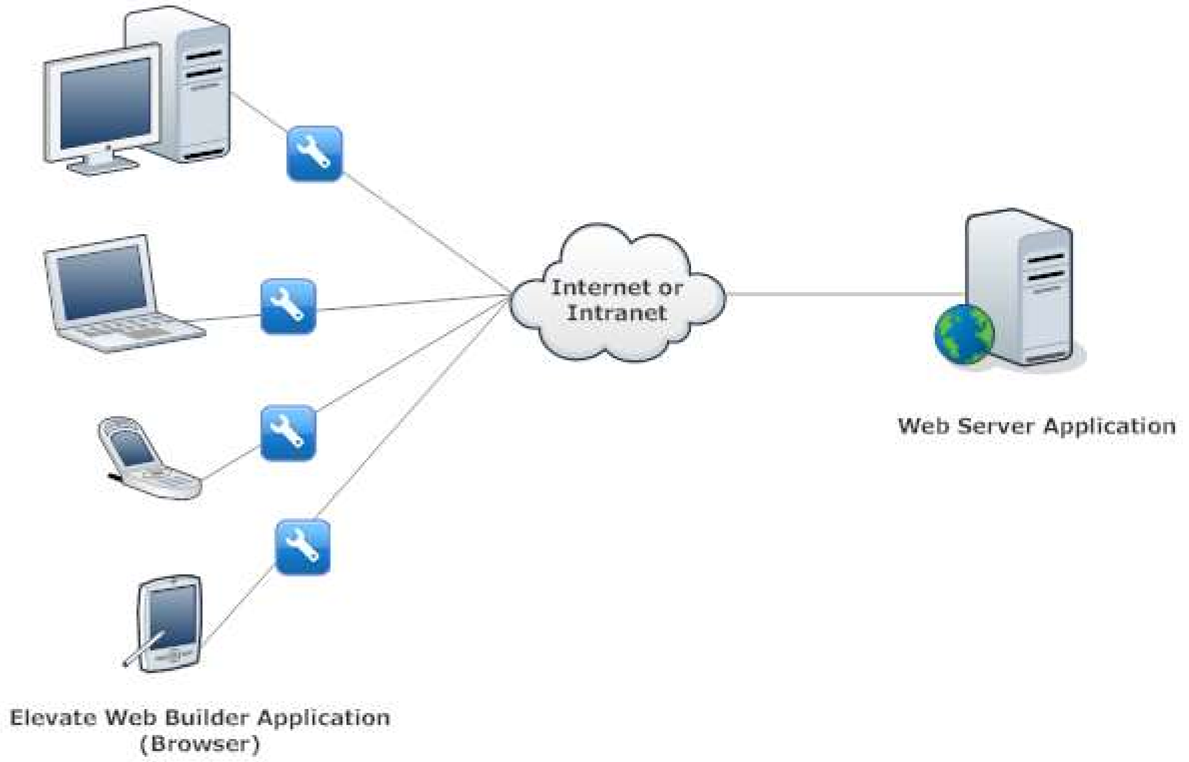
Mobile Browsers

Most mobile devices today use the WebKit web browser engine as the engine for their web browser. This is the case with both Android and iOS devices. Elevate Web Builder applications will work properly with any device that uses the WebKit web browser engine. Some other devices may use embedded versions of the Opera web browser engine, and Elevate Web Builder applications will work properly with those devices as well.

1.2 General Architecture

Elevate Web Builder allows developers to easily create rich, fully-functional web browser applications that use 100% standard browser technologies and don't require any external browser plugins or layers.

At run-time, an Elevate Web Builder application has the following architecture:



Elevate Web Builder Architecture

Elevate Web Builder's IDE/compiler only produces web browser applications, and does not produce the web server application. However, included with the product is the Elevate Web Builder Web Server, which is the same internal web server used in the IDE for running applications. You can create native web server application modules using Embarcadero RAD Studio and Delphi by using the Elevate Web Builder Web Server. Please see the following topics for more information on the Elevate Web Builder Web Server:

Starting the Web Server
Configuring the Web Server
Web Server Request Handling
Creating Web Server Modules

You are required to create a web server application if you want to accept requests from the web browser application, such as those used for loading datasets, committing transactions, or executing custom requests to the web server application. Please see the Database Architecture and Server Request Architecture topics for more information on how databases and server requests work in Elevate Web Builder applications.

Core Language

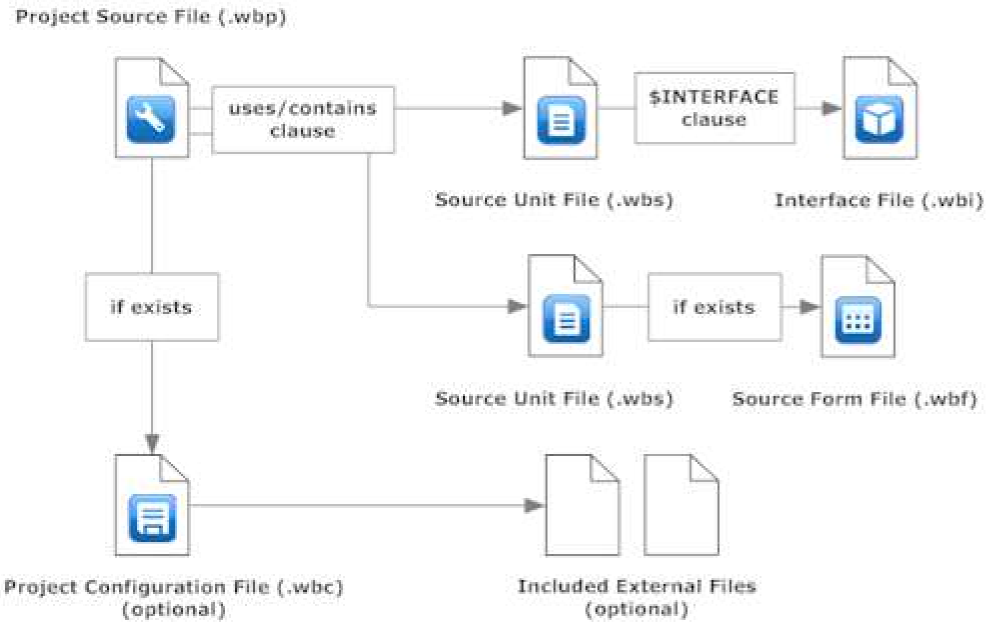
Elevate Web Builder uses an Object Pascal language dialect for its source code that is very close to the Object Pascal language used by the RAD Studio and Delphi development environments from Embarcadero Technologies. Object Pascal was chosen as the language because it is a very easy language to learn due to its very English-like keywords, and because it is highly-structured and strongly-typed, making compiled applications highly-resistant to easily-avoided run-time type or class definition errors. For more information on the language in Elevate Web Builder, please see the Language Reference section of the manual.

Integrated Development Environment (IDE)

The IDE in Elevate Web Builder is also modeled after the RAD Studio and Delphi IDEs from Embarcadero Technologies, and is specifically designed to facilitate rapid application development (RAD). Rapid application development is a development process that allows a developer to quickly proceed from an application design to a fully-functional application by tightly integrating the design portion of application development with the coding/compilation/deployment portion of development. Please see the Using the IDE topic for more information on the layout and usage of the IDE.

1.3 Application Structure

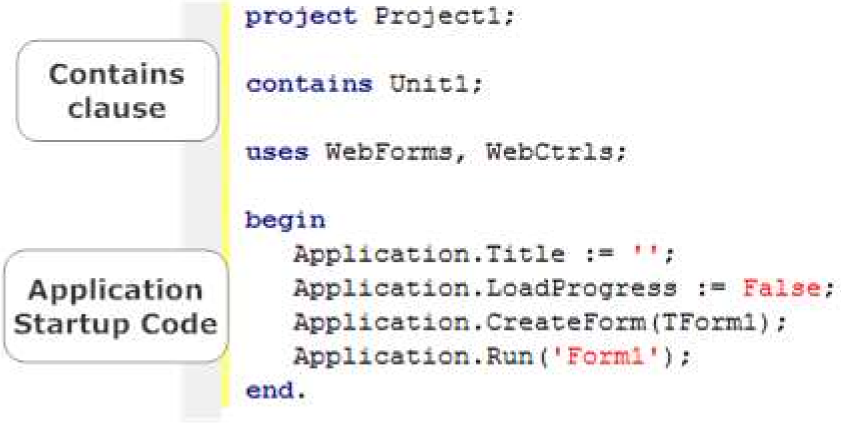
An Elevate Web Builder application is structured as follows:



Elevate Web Builder Application Structure

An application can be either visual or non-visual. The main difference between visual projects and non-visual projects is the way that the IDE generates code in the project source file as forms/source units are added and removed from the project.

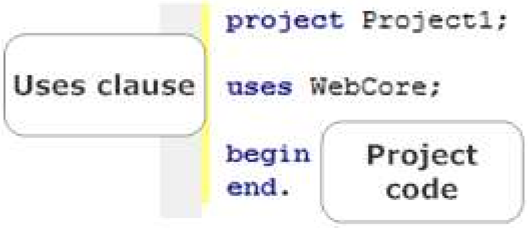
The project source file (.wbp) looks like the following for a visual application:



```
project Project1;

contains Unit1;

uses WebForms, WebCtrls;

begin
   Application.Title := '';
   Application.LoadProgress := False;
   Application.CreateForm(TForm1);
   Application.Run('Form1');
end.
```

Project Source File

The project source file (.wbp) looks like the following for a non-visual application:



```
project Project1;

uses WebCore;

begin
end.
```

The IDE and compiler use the **contains** clause in the project source file to determine which source units are part of the actual project (as opposed to simply being referenced in a source unit that is part of the project). This is important for several dialogs in the IDE that present a list of source units or forms for selection, as well as the Project Manager.

> **Note**
> By default, the IDE does not create a contains clause for non-visual projects because non-visual projects don't have any additional project units when first created. However, you can add units to the project using the project manager, and they will appear in the contains clause of the project source file.

Project Configuration File

The project configuration file (.wbc) is an .ini file with the same root name as the project source file that contains the project settings for the current project. If it does not exist, then it is automatically created by the IDE. The settings stored in this file include:

- IDE layout settings (open windows, panel positions/sizes)
- Compilation settings (search paths, output paths, compression)
- External files (JavaScript, images)
- Deployment settings (FTP settings, copy settings)

Form Files

Forms are associated with a specific source unit by the existence of a .wbf form file with the same root name as the .wbs source unit. Form files are JSON files that contain information about all components contained within a form and all non-default published property values assigned to the components.

Control Interface Files

Control interfaces are associated with an application or the component library via this compiler directive:

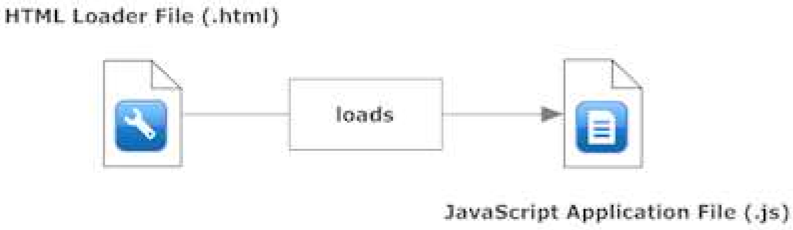
```
{$INTERFACE <ControlInterfaceFileNameRoot>}
```

Control interfaces are JSON files that contain information about the various visual states of a control interface class. Each state is represented by one or more UI elements that correspond to the UI elements created by a control class. Please see the Control Interfaces topic for more information on the architecture of control interfaces.

1.4 Compiling Applications

When you compile an application, the Elevate Web Builder uses the project's compiler search paths along with the component library search paths to determine where to look for units and control interface files. Please see the Modifying the Project Options and Modifying Environment Options topics for more information on modifying these search paths.

Elevate Web Builder compiles the application source code (Object Pascal) into a 100% JavaScript application that will run in any modern browser. During compilation of an application, the compiler emits the following files:



Elevate Web Builder Runtime Application Structure

The HTML loader file contains all of the control interface files, form files, and database files in special tags in the header of the file.

An Elevate Web Builder application is typically loaded in a web browser via a URL that includes the HTML loader file for the application. Once the loader file is loaded in the web browser, the following steps occur:

- The HTML loader file loads the application's .js (JavaScript) file, which causes the browser to compile the JavaScript and prepare the execution environment.

- A special JavaScript loader function is called that initializes the application and starts execution.

- Any control interfaces are loaded from the special tags in the HTML loader file.

- Any auto-create forms and/or databases are created. During creation, the associated form or database files are loaded from the special tags in the HTML loader file.

- If the web browser navigates away from the current URL, or if the web browser refreshes the current URL, then a special JavaScript unloader function is called that cleans up all allocated resources and terminates the application.

Elevate Web Builder applications are designed to be loaded and then stay loaded until they are exited. They are not "page-oriented" like many web application or general web sites. You can, however, display HTML pages within an application by using the TBrowser control that is provided as part of the component library.

The JavaScript file that is emitted by the compiler can be compressed, making the size of the resultant application much smaller. As a side-effect of the compression, the resultant JavaScript source code is also heavily obfuscated and virtually unreadable, which is desirable for many applications that wish to protect their source code. Please see the Modifying the Project Options topic for more information on compression.

1.5 Component Library

Elevate Web Builder includes a complete component library for use with both your visual and non-visual web applications. The component library is a separate design-time application that is automatically loaded and compiled by the IDE during startup. You can add and remove components from the library in the IDE, and those changes will persist for any subsequent IDE usage. You can also rebuild the component library at any time, which is useful for situations where an existing component or control is modified, and you wish to have those changes reflected immediately at design-time.

> **Note**
> The component library is the foundation for all design-time visual designers: you cannot work with visual application projects and forms unless the component library has been successfully loaded and compiled.

The TComponent class is the base class for all components and contains all core functionality for component ownership and notification. It inherits from the TPersistent class, so any TComponent-descendant class can automatically load itself from a JSON input string.

The locations of the component library source units are automatically included in the compiler search path during the compilation of projects, but after the project's defined compiler search paths. Please see the Modifying Environment Options topic for more information on modifying the component library search paths used by the compiler.

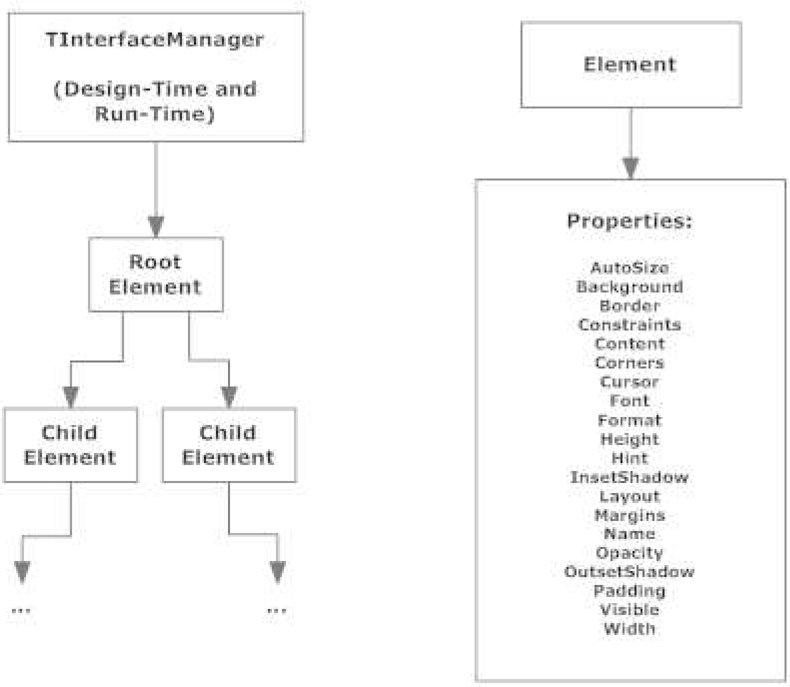
The following source units make up the core of the runtime and component library:

Source Unit	Description
WebDOM	This source unit contains all external declarations for the web browser DOM (Document Object Model) classes, functions, procedures, and variables. You can use this unit to manipulate the browser DOM directly at run-time.
WebDesign	This source unit contains just a few class declarations for use with the design-time environment in the IDE.
WebCore	This source unit contains core functions/procedures and classes. It does not contain any visual controls so it can be used in non-visual applications.
WebUI	This source unit contains the interface manager and all of the base UI functionality for both design-time and run-time.
WebCtrls	This source unit contains the base controls and functionality.
WebForms	This source unit contains the application, surface, and core form/dialog controls.
WebData	This source unit contains the database and dataset components.
WebHTTP	This source unit contains the server request components.

In addition to these units, there are many other units that make up the rest of the component library. Please see the Component Reference section of this manual for detailed documentation about the units, classes, and types in the Elevate Web Builder component library.

1.6 Visual Applications

The interface of a visual Elevate Web Builder application has the following structure:



Elevate Web Builder Visual Application Interface

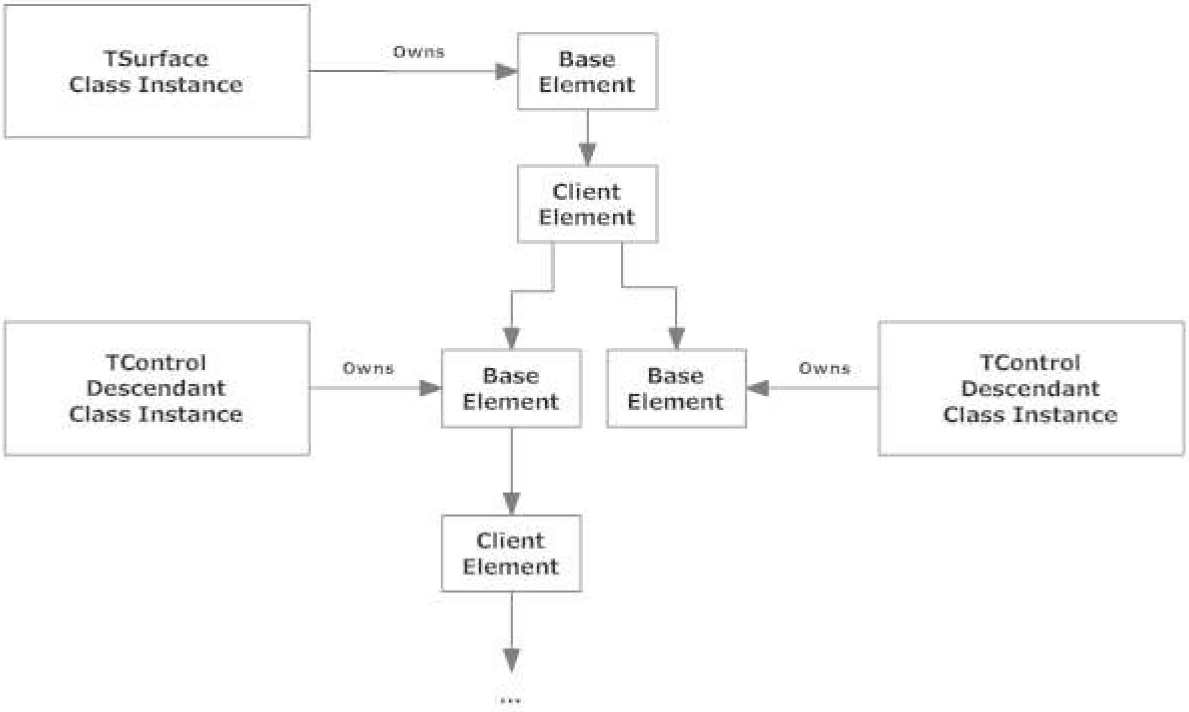
A visual application uses a global instance of the TInterfaceManager class called InterfaceManager to manage all aspects of the user interface. The TInterfaceManager class and all related classes, types, and functions/procedures are declared in the WebUI unit included with the standard component library. All interface elements are represented by instances of the TElement class (or a descendant class).

At design-time, the root element managed by the interface manager represents the base element for the active form instance in the form designer. At run-time, the root element is the base element for the global instance of the application surface. The application surface wraps the body element in the browser tree of elements, and is the ancestor container for all controls at run-time.

Note
At run-time, all elements are wrappers around browser elements. Except for the body element, these browser elements are owned by the TElement class instances of the corresponding interface elements.

Controls and Interface Elements

In Elevate Web Builder, controls are simply wrappers around a base element that is the container for all elements that are created and owned by the control itself, or assigned as child elements:



Elevate Web Builder Controls and UI Elements

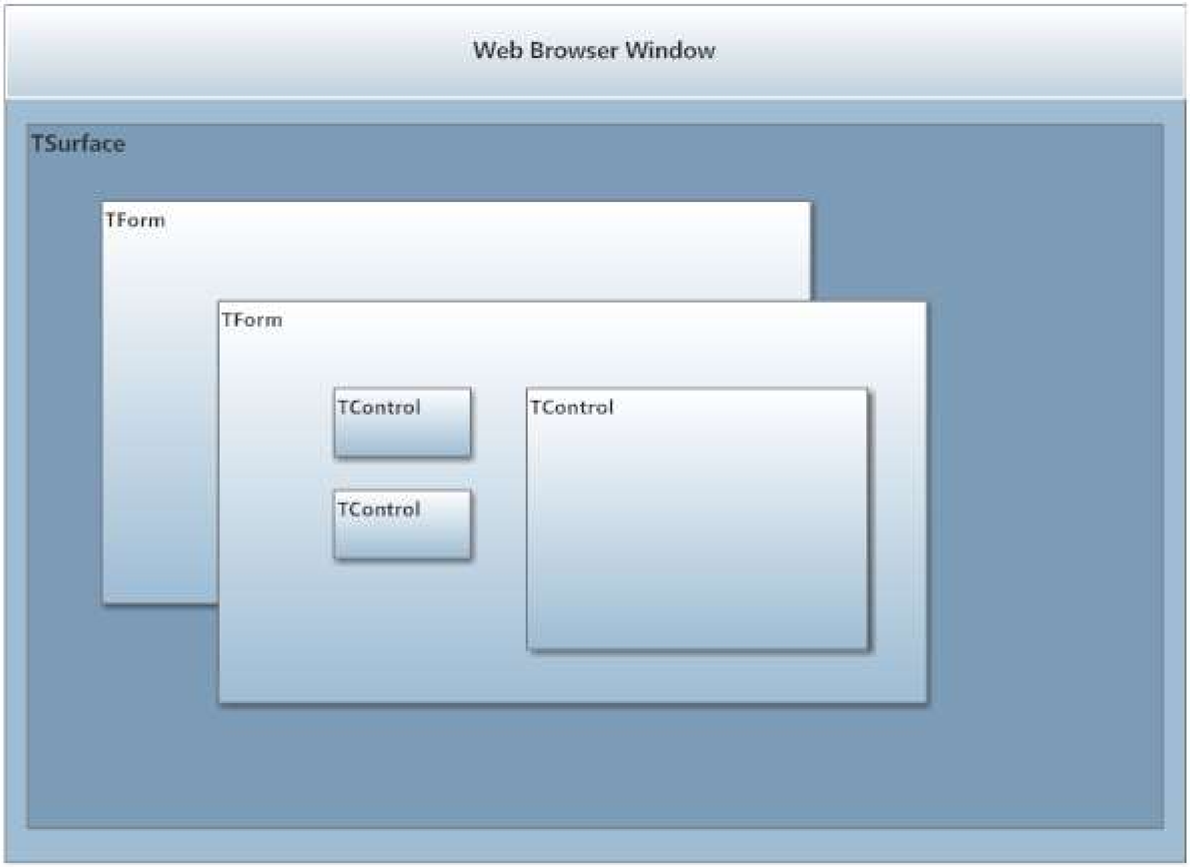
Many controls never use more than the base element, while others such as grid controls require many elements in order to provide the necessary functionality in the control.

Controls create both their base element and any other owned elements by calling the TInterfaceManager CreateElement method. Controls, and the elements that they own or parent, can be moved to different parts of the user interface tree of elements by modifying the TControl Parent property.

Because controls are simply wrappers around interface elements, it is up to the control class to determine what aspects of the owned element(s) is/are exposed at design-time and at run-time via properties. For example, a container control may wish to expose the background property of the base element so that the developer can modify the background of the control.

Core Application Components

A visual Elevate Web Builder application has the following structure in the web browser:



The visual application functionality in Elevate Web Builder contains several core components, all residing in the WebForms and WebCtrls units in the standard component library.

TControl
The TControl component is the base class for all controls and forms. It contains all core functionality for control iteration, dimension and layout management, and display control. You can find the TControl component in the WebCtrls unit.

TApplication
An instance of the TApplication component is automatically created when the WebForms unit is initialized, and is available as the global Application variable (also in the WebForms unit). The TApplication component allows the developer to manage the browser surface via the Surface property, as well as various aspects of the application such as the application title and the properties of the browser viewport.

TSurface
An instance of the TSurface component is automatically created by, and as part of, the global TApplication instance and, as noted above, is available via the TApplication Surface property. You can access the active form via the ActiveForm property.

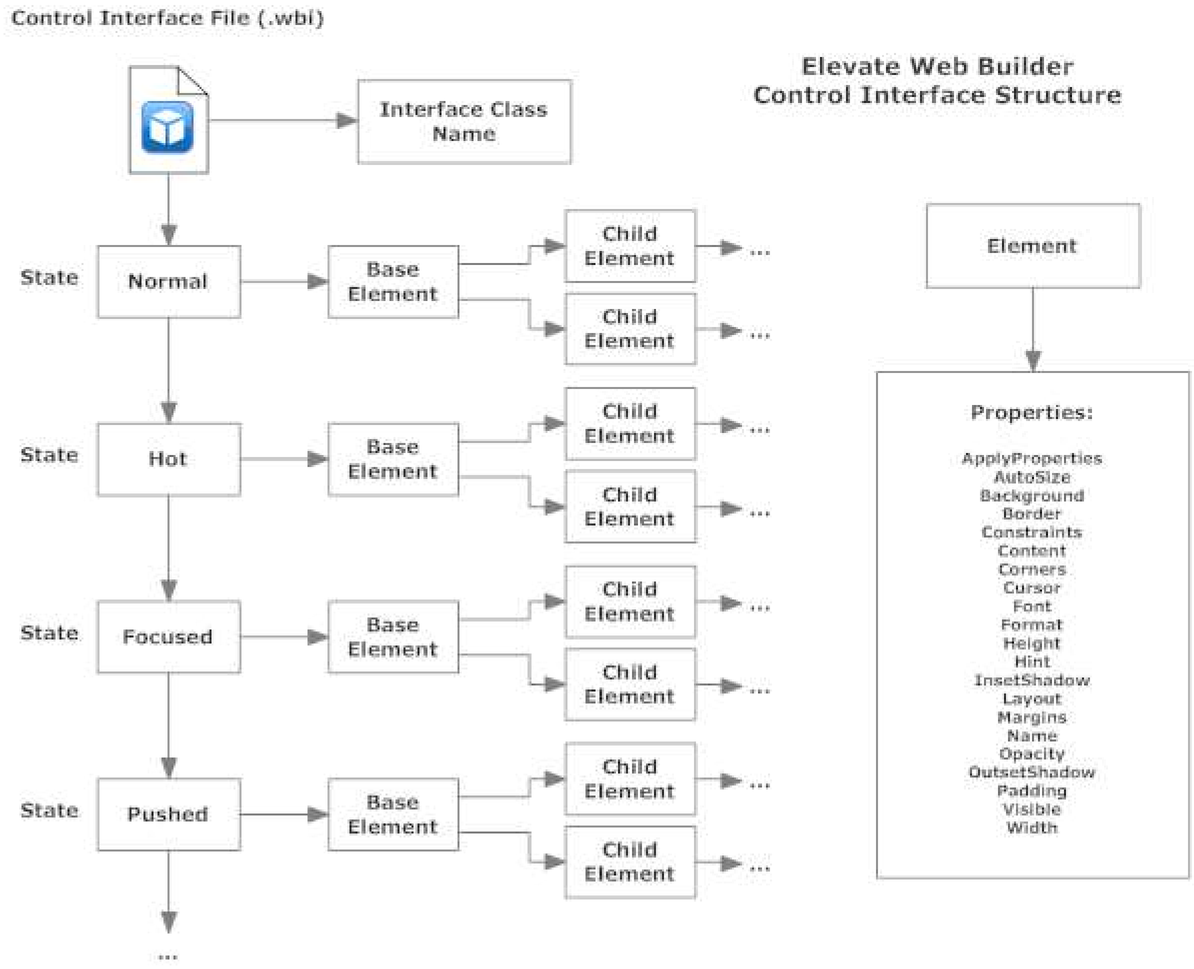
TForm
The TForm component encapsulates a form in an Elevate Web Builder visual application. Forms are the container controls in which all other visual controls reside. Forms can be designated as auto-create in a project and the IDE will automatically add the appropriate code to the program source of the project for creating the forms at application start. The first form in the list of auto-create forms is considered the main form of the application. The TForm class can be found in the WebForms unit.

TDialog

The TDialog component encapsulates a dialog in an Elevate Web Builder visual application. Dialogs differ from normal forms in that they contain additional interface elements such as a caption bar and a close button. Also, dialogs are normally shown modally using the TFormControl ShowModal method, although this is not a requirement. The TDialog class can also be found in the WebForms unit.

1.7 Control Interfaces

Control interfaces are JSON files that are used to describe the visual appearance of a control. You can create and modify control interfaces using the Control Interface Editor. The structure of a control interface is as follows:



As mentioned in the Application Structure topic, control interfaces are included in a source unit via the $INTERFACE compiler directive and, as mentioned in the Compiling Applications topic, control interfaces are automatically stored in the application's HTML loader file by the compiler, and are automatically loaded at application startup by the global TInterfaceManager instance called InterfaceManager in the WebUI unit.

> **Note**
> The name of a control interface file will not necessarily correspond to the interface class name defined within the file. The standard control interface files shipped with Elevate Web Builder use the same name for both, but this is only a convention and not a requirement.

TControl Interface Functionality

As discussed in the Visual Applications topic, every TControl descendant class will create one or more TElement class instances to handle the various interface elements in the control. By default, the TControl class automatically creates an element with the name "Base". All element instances should have a unique name within the context of a control class.

> **Note**
> There are several standard interface element names defined in the interface section of the WebCtrls unit that should be used with any controls, if possible. For example, controls should normally always call any client element "Client". This isn't a requirement, but it does help keep any 3rd party controls standardized.

In addition, every TControl descendant class is always in a particular interface state, represented by the protected TControl InterfaceState property. Each TControl can modify its InterfaceState property to affect the layout and display properties of the element instances in the control. By default, the base TControl class automatically handles most interface state changes. The standard interface states, and how they are triggered, are detailed below:

Interface State	Trigger Condition
Normal	This is the default state for controls, and is triggered during control initialization.
Disabled	Triggered when the TControl Disabled property is set to True.
Hot	Triggered when the mouse is moved over the bounds of the control.
Focused	Triggered when the control obtains focus.
Pushed	Triggered when the left mouse button is pressed within the bounds of the control.
Minimized	Triggered when the control (normally a container control) is minimized.
ReadOnly	Triggered when the TBindableControl ReadOnly property (protected) is set to True.
Error	Triggered when the TBindableControl Error property (protected) is set to True.

There are several other control-specific states that are used by the standard component library component classes. A control developer is free to use any interface state name that they wish, but the standard interface states should not be used for purposes other than their intended use.

How Control Interfaces are Applied to a Control

Internally, control interfaces are applied to the interface elements of controls using the TInterfaceManager ApplyInterface method. This method uses several key pieces of information to determine how to apply an interface to a control and the interface elements that it owns:

- The interface class name
- A specified interface state

- The interface element names

When a control assigns a new value to the protected TControl InterfaceState property and the TControl class calls the TInterfaceManager ApplyInterface method, the method will look up the control interface class name in the list of available control interface class names loaded into the interface manager.

The control interface class name is provided by the TControl GetInterfaceClassName method:

```
function GetInterfaceClassName: String; virtual;
```

It can be used to return any string that represents the control interface class name that the control wants to apply when the interface state changes. Normally, the value returned here is simply the class name for the control. For example, the TEdit control GetInterfaceClassName method implementation looks like this:

```
function TEdit.GetInterfaceClassName: String;
begin
   Result:=TEdit.ClassName;
end;
```

However, you are **not** forced to use the control's class name as the interface class name. Sometimes it might be necessary to use the control's class name combined with other string values to create a dynamic interface class name. For example, the TScrollBar class uses the orientation of the scroll bar to compute a dynamic interface class name:

```
function TScrollBar.GetInterfaceClassName: String;
begin
   case FOrientation of
      soVertical:
         Result:=TScrollBar.ClassName+VERTICAL_CLASS_NAME;
      soHorizontal:
         Result:=TScrollBar.ClassName+HORIZONTAL_CLASS_NAME;
      else
         Result:='';
      end;
end;
```

Once the control interface class name is found by the interface manager, the control interface states are searched for the value assigned to the InterfaceState property. If a matching state is found in the control interface, then the properties of the various elements in the control interface state are applied to the element instances contained within the control class instance whose interface state is being changed. This property application process is also done by matching the names of the elements in the control interface to the names of the element instances in the control class instance.

Note
If an interface state is specified that does not exist, then nothing occurs and the visual appearance of the control will not change. If one or more element names in the control interface state do not match the names of the element instances created by the control class instance, then the properties of those elements will not be applied.

If an interface state is being assigned to a control class instance for the first time (InterfaceState property is blank), then the interface manager will simply assign all properties of the control interface elements to the matching element instances contained within the control class instance. If an interface state has already been assigned to the control class instance, then the interface manager will only apply the properties specified via the control interface element's ApplyProperties property. The ApplyProperties property of a control interface element is a set of Boolean values that mirror the control interface element properties. If a property is set to True in the ApplyProperties property, then it will be assigned to the element instance when the interface state is applied.

Note
If you specify that a property should be applied in any state, then you should normally also specify that the property should be applied in all other states in the control interface. Failure to do this will result in certain properties becoming "sticky" and not reverting to a known state as the interface state of the control changes. For example, suppose that you have created a TBorderButton control that you want to show a different color border when the mouse hovers over the control and the interface state changes to "Hot". In such a case, you'll need to be sure that the ApplyProperties.Border property is set to True for the Base element defined in all states in the control interface, including the standard Normal state.

Customizing Control Interfaces

Elevate Web Builder ships with a complete set of standard control interfaces located in the \interfaces subdirectory under the main installation directory. Because the compiler uses the standard project and component library search paths to find both source units and control interfaces, you can make copies of the standard control interfaces, place them in a new directory, and then modify the project's compiler search paths so that the new directory is included. Please see the Modifying Project Options topic for more information on modifying the project's compiler search paths. After that point, any modifications to the control interfaces in this new directory will be used instead of the standard control interfaces. Please see the Modifying a Control Interface topic for more information on how to modify control interfaces using the control interface editor.

By default, the **Automatically load custom control interfaces in project search paths** option in the Environment Options dialog is enabled. This means that the IDE will automatically load any custom control interface files located in the project's compiler search paths whenever a project is opened in the IDE. When checking to see if a control interface has been customized, the IDE compares the path of the default control interface file used with the component library (based upon the Library Search Paths setting on the Component Library page) with the path of any control interfaces with the same file name present in the project's compiler search paths. If a match is found, then the control interface file found in the project's compiler search paths is loaded into the IDE and used with the project's form designers. After the project is closed, the default control interfaces are reloaded.

1.8 Icon Library

Icons are small, rectangular images/symbols that are referenced and displayed using controls such as the TIcon component. Elevate Web Builder uses a special control interface class called TIconLibrary to embed icons in a visual application. The TIconLibrary control interface is stored in the TIconLibrary.wbi interface file included with the standard control interfaces provided with Elevate Web Builder. The default TIconLibrary control interface contains several default icons. You can use them as templates for any additional icons by specifying their name as the icon (state) to copy when adding a new icon (state). You can make a copy of the TIconLibrary.wbi interface file and place it in your project source directory in order to customize the icons for your application. Please see the Opening the Icon Library for more information on how to save a copy of the default TIconLibrary control interface so that it can be customized.

Supported Icon Types

Elevate Web Builder supports two different kinds of icons in the icon library: raster image icons (PNG) and font icons. All icons used in the Elevate Web Builder component library use font icons. Font icons are preferable to image icons because font icons are vectors, not raster images. This means that they can be resized without losing any clarity, which is very important with very high display resolutions such as those used with the Retina displays available on Apple devices. Raster images, on the other hand, tend to look blurry as the image pixels are stretched and compressed to fit the current scale of the browser and the underlying display resolution. Even if you aren't targeting high-resolution displays, you will want your icons to look crisp and clear if the user zooms in/out using the built-in scaling available in most browsers. Another important consideration is that font icons are much smaller, overall, than the equivalent raster images. Finally, the fill color of font icons can be changed like any other font, whereas the colors of raster images are fixed. However, raster images allow for multiple colors in icons, which is not something that is currently supported with font icons.

Icon Fonts

Elevate Web Builder ships with an icon font called **EWBIcons** in both OpenType format and WOFF (web font) format. The OpenType version of the EWBIcons icon font is automatically installed during the Elevate Web Builder installation, and both formats are available in the \fonts subdirectory under the main installation directory.

The EWBIcons icon font is a trimmed-down version of the fantastic open source icon font called **Font Awesome** available here:

Font Awesome

With the EWBIcons icon font, the social media and brand icons were stripped from the original Font Awesome font in order to conserve space, and a few icons were added to support dataset toolbar navigation icons and the Elevate Web Builder "tool" logo icon. Because of this, the font name had to be changed from "FontAwesome" to "EWBIcons" in order to avoid any conflicts.

By default, all projects will include the EWBIcons icon font during compilation, and will embed it in the HTML loader file for the application. This can be behavior can be changed via the Compilation page of the Project Options for each project.

Defining Icons

Icons are represented in the states of the control interface, with the name of the icon corresponding to the state name. There are no limits to how many icons (states) one can define in the control interface. However, there are some rules that must be followed in order to allow the icons to appear correctly. The following lists the rules for both image icons and font icons:

Image Icons

- The base element for each icon (state) **must** be named "Icon".
- The "Icon" element's ApplyProperties Background property should be set to True.
- The icon image should be assigned using the "Icon" element's Background Image Name property.
- The icon image itself should be sized according to your needs, but as a rule do not use icons larger than 256x256 pixels. The one exception to this rule is when defining an animated icon. Defining animated icons is discussed below.
- The icon image should normally be positioned as ptCenterCenter using the "Icon" element's background image PositionType property. The one exception to this rule is when defining an animated icon. Defining animated icons is discussed below.
- The icon image should be set to not repeat by setting the "Icon" element's background image RepeatStyle property to rsNone.

Font Icons

- The base element for each icon (state) **must** be named "Icon".
- The icon's base "Icon" element should have a child element named "FontIcon".
- The "FontIcon" element's ApplyProperties AutoSize, Content, Font, FontColor, FontSize, Layout, and Padding properties should be set to True.
- The "FontIcon" element's AutoSize property should be set to True.
- The "FontIcon" element's Font Name property should be set to "EWBIcon", or the name of another icon font that you wish to use.
- The "FontIcon" element's Font Color and Size properties should be set according to your needs. The Font Style property is not normally used with font icons.
- The "FontIcon" element's Layout Position property should normally be set to lpCenter, but you can use any layout that you wish for the font icon.
- The "FontIcon" element's Padding property can be used to adjust the padding around the font icon. This is useful in cases where the font icon is getting cut off due to leading/trailing measurement issues with the font.

Defining Animated Icons

Animated icons are image icons whose background image isn't a small square or rectangle, but is instead a single image containing many different animation frames oriented in a single horizontal or vertical direction. These icons are referenced and displayed using controls such as the TAnimatedIcon component. In addition to the above rules for normal icons, there are also some rules that must be followed in order to allow animated icons to appear correctly:

- Contrary to normal icons, an animated icon image should **always** be positioned as ptSpecified using the "Icon" element's background image PositionType property.

Controls such as the TAnimatedIcon control use the background image's BeginAnimation method to animate the background image's Left or Top property (depending upon the orientation passed to the method), and the CancelAnimation method to stop any background image animation.

Loading Icons from Code

At both design-time and run-time, a global instance of the TIconLibrary class called IconLibrary is created that manages this special icon library control interface. The TIconLibrary class and the global IconLibrary instance can be found in the WebUI unit. Controls like the TIcon control use this global instance to retrieve the list of available icons using the GetIconNames method, as well as apply an icon to one or more of its owned interface elements using the ApplyIcon method.

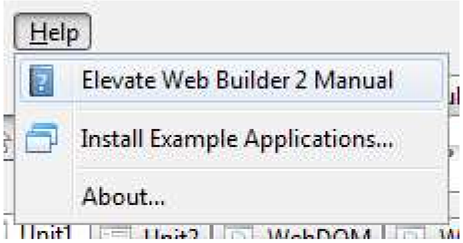
1.9 Accessing Help

Elevate Web Builder includes a complete online manual in the IDE that provides language and component references, as well as tutorials and information on how to use the product to create great web browser applications.

Accessing the Online Manual

Use the following steps to access this manual in the IDE:

- Click on the **Help** option in the main menu. The Help menu will open:

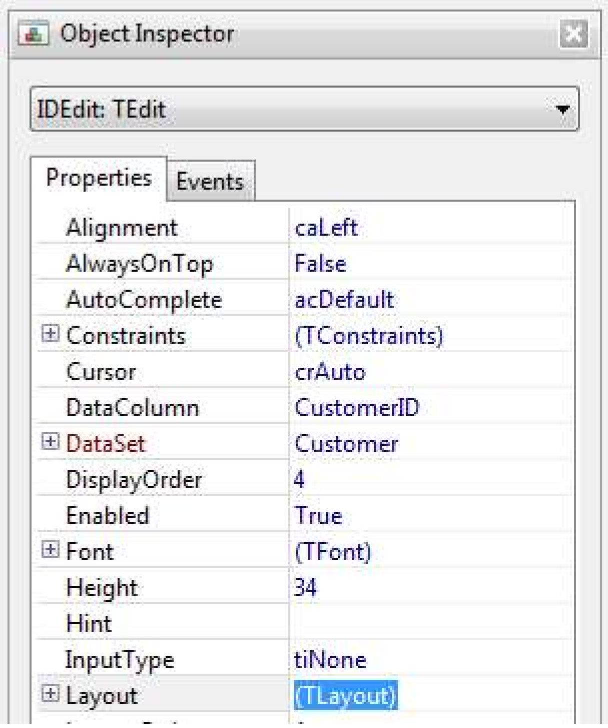


- Click on the **Online Manual** option in the Help menu. This will cause the help browser page to open in the IDE.

Context-Sensitive Help in the Object Inspector

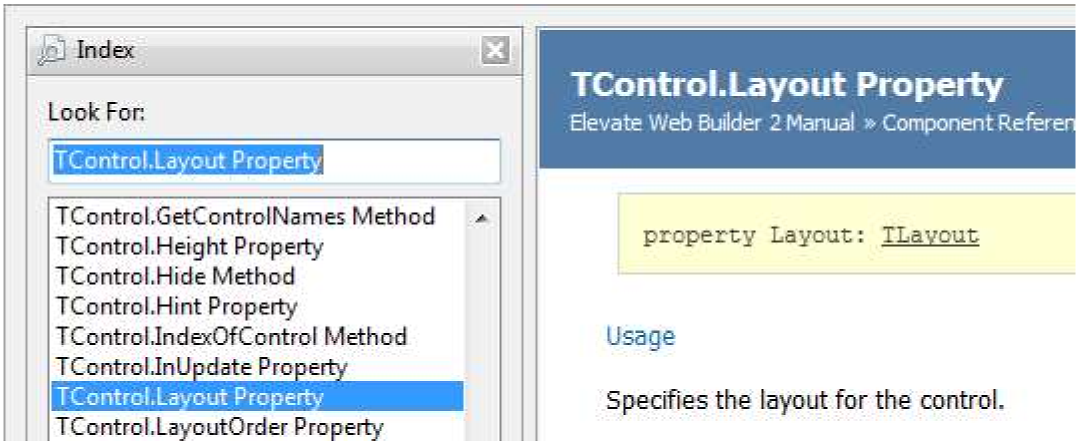
Use the following steps in order to obtain context sensitive help in the object inspector:

- Click on the desired property in the object inspector and hit the F1 key.



Alignment	caLeft
AlwaysOnTop	False
AutoComplete	acDefault
⊞ Constraints	(TConstraints)
Cursor	crAuto
DataColumn	CustomerID
⊞ DataSet	Customer
DisplayOrder	4
Enabled	True
⊞ Font	(TFont)
Height	34
Hint	
InputType	tiNone
⊞ Layout	(TLayout)

- The help browser will open with all matching keywords highlighted. The topic that corresponds to the first matching keyword will be displayed in the help browser.



TControl.Layout Property

TControl.GetControlNames Method
TControl.Height Property
TControl.Hide Method
TControl.Hint Property
TControl.IndexOfControl Method
TControl.InUpdate Property
TControl.Layout Property
TControl.LayoutOrder Property

TControl.Layout Property

Elevate Web Builder 2 Manual » Component Referen

```
property Layout: TLayout
```

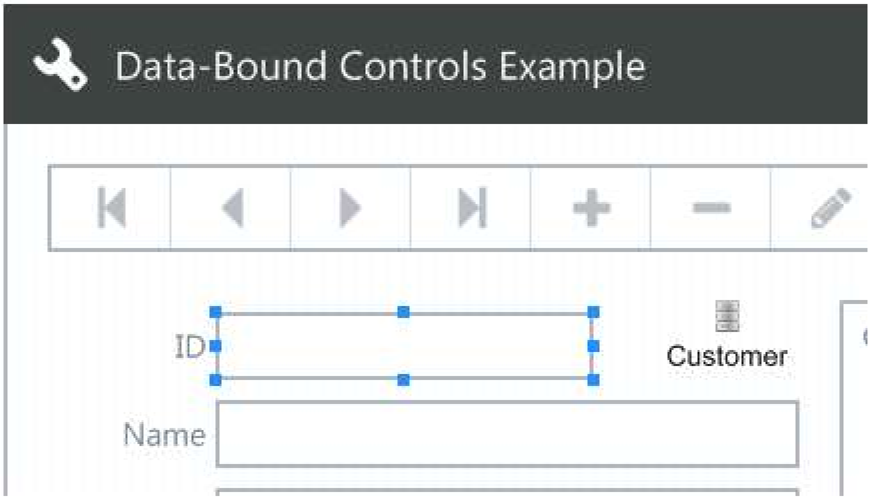
Usage

Specifies the layout for the control.

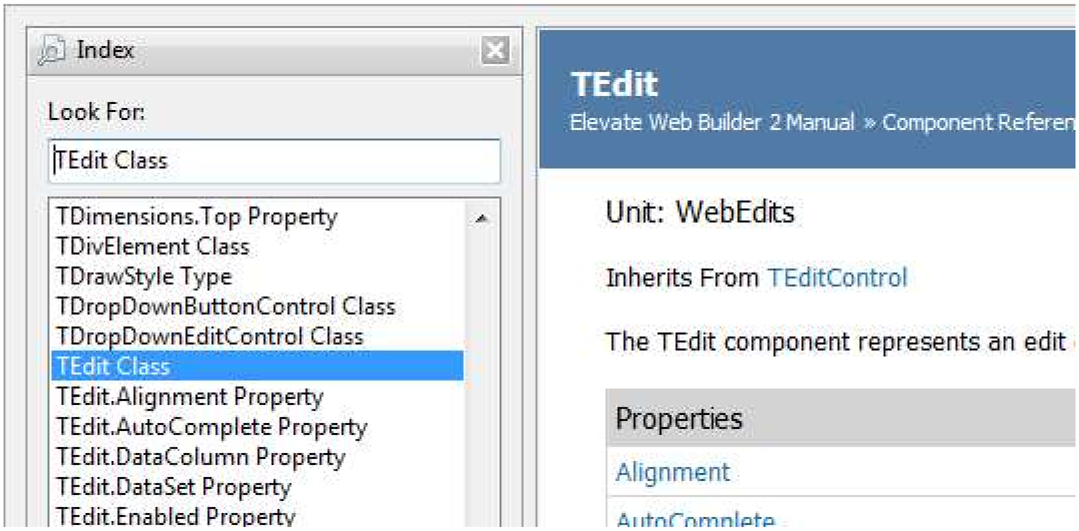
Context-Sensitive Help in the Form and Database Designers

Use the following steps in order to obtain context sensitive help in the form and database designers:

- Click on the desired control or component in the form or database designer and hit the F1 key.



- The help browser will open with all matching keywords highlighted. The topic that corresponds to the first matching keyword will be displayed in the help browser.



Look For:

TEdit Class

TDimensions.Top Property
TDivElement Class
TDrawStyle Type
TDropDownButtonControl Class
TDropDownEditControl Class
TEdit Class
TEdit.Alignment Property
TEdit.AutoComplete Property
TEdit.DataColumn Property
TEdit.DataSet Property
TEdit.Enabled Property

TEdit

Elevate Web Builder 2 Manual » Component Referen

Unit: WebEdits

Inherits From TEditControl

The TEdit component represents an edit

Properties

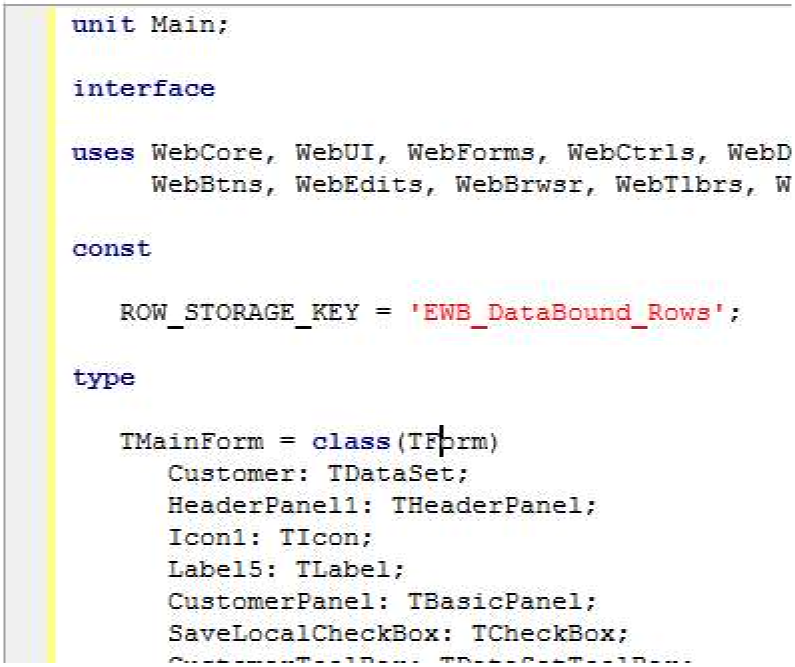
Alignment

AutoComplete

Context-Sensitive Help in the Code Editor

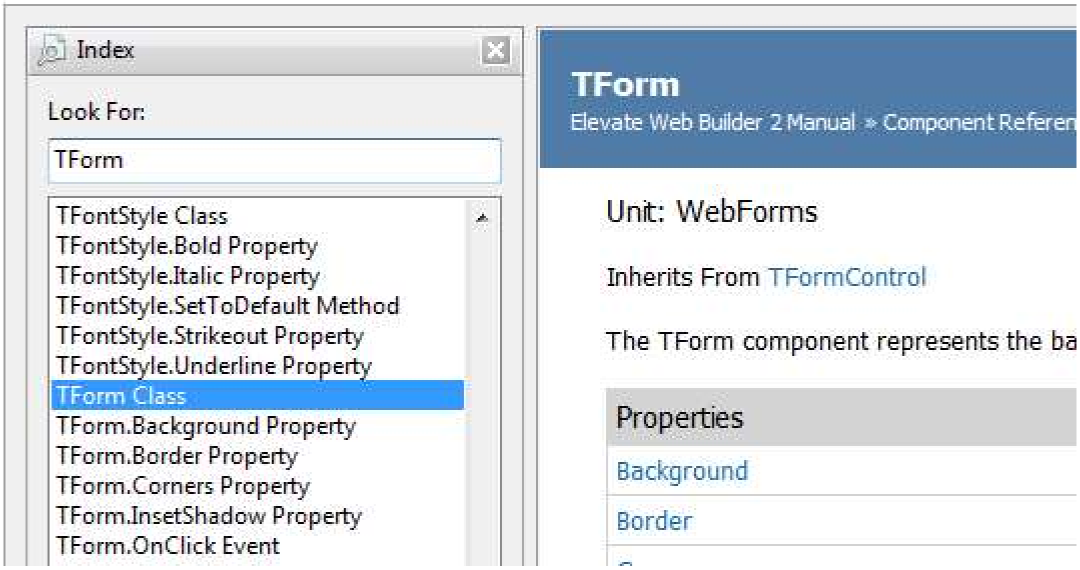
Use the following steps in order to obtain context sensitive help in the code editor:

- Click on the desired keyword or identifier in the code editor and hit the F1 key.



```
unit Main;

interface

uses WebCore, WebUI, WebForms, WebCtrls, WebD
     WebBtns, WebEdits, WebBrwsr, WebTlbrs, W

const

   ROW_STORAGE_KEY = 'EWB_DataBound_Rows';

type

   TMainForm = class(TForm)
      Customer: TDataSet;
      HeaderPanel1: THeaderPanel;
      Icon1: TIcon;
      Label5: TLabel;
      CustomerPanel: TBasicPanel;
      SaveLocalCheckBox: TCheckBox;
```

- The help browser will open with all matching keywords highlighted. The topic that corresponds to the first matching keyword will be displayed in the help browser.



Look For:

TForm

TFontStyle Class
TFontStyle.Bold Property
TFontStyle.Italic Property
TFontStyle.SetToDefault Method
TFontStyle.Strikeout Property
TFontStyle.Underline Property
TForm Class
TForm.Background Property
TForm.Border Property
TForm.Corners Property
TForm.InsetShadow Property
TForm.OnClick Event

TForm
Elevate Web Builder 2 Manual » Component Referen

Unit: WebForms

Inherits From TFormControl

The TForm component represents the ba

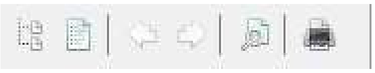
Properties

Background

Border

Using the Help Browser

The help browser navigation toolbar can be found at the bottom of the help browser window:



The navigation buttons are:

Displays the contents tree on the left-hand side of the help browser.

Displays the keywords index on the left-hand side of the help browser.

Allows you to navigate backward and forward from the current topic to the previous topic or next topic viewed.

In addition you can use the following toolbar buttons:

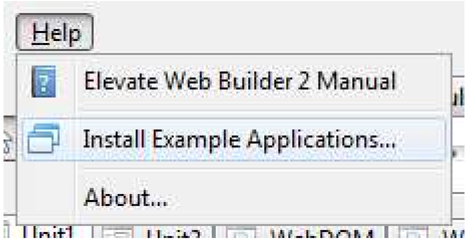
Searches for text within the current topic.

Prints the current topic to the desired output device.

1.10 Example Applications

Elevate Web Builder includes several example applications, which are detailed below. By default, these example applications are installed into the \examples subdirectory under the main installation directory for Elevate Web Builder. However, you should **not** try to load or compile the example projects from this location. This is because Elevate Web Builder is normally installed under the \Program Files directory structure under Windows, which will cause the Elevate Web Builder compiler to encounter errors when trying to create the proper output directories and files during the emitting phase of compilation. Rather, you should use the following steps to install the example applications in the documents folder for the current user account:

- Click on the **Help** option in the main menu. The Help menu will open:



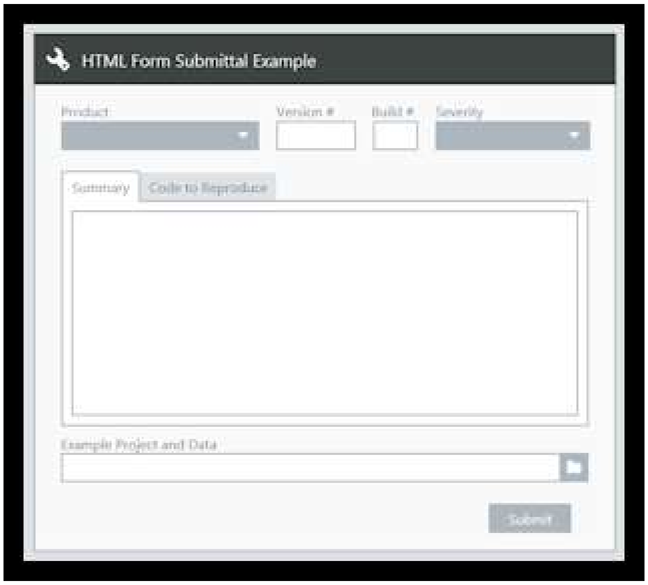
- Click on the **Install Example Applications..** option in the Help menu. This will start the process of copying the example applications to the following folder for the current user account:

```
My Documents\Elevate Web Builder 2\Projects
```

Note
All of these example applications require that the internal web server in the IDE is started and running, or you will get errors when trying to run them in the IDE (or from any browser). Also, do not attempt to run any of the example applications directly from the file system in the IDE (or from any browser) or you will also get errors. Please see the Running a Project topic for more information on running projects in Elevate Web Builder.

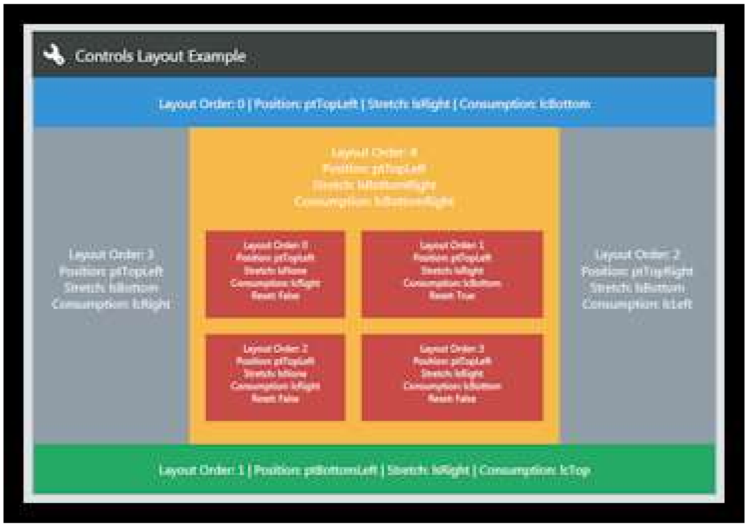
In addition, all of the example projects use the default control interfaces. Please see the Control Interfaces topic for more information on how control interfaces work.

HTML Form Submittal Example



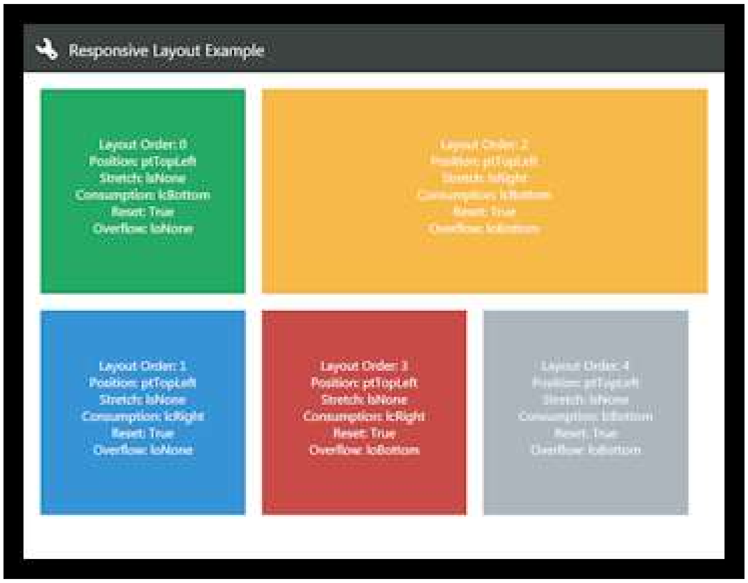
After installation, the formsubmit.wbp example project will be located in the My Documents\Elevate Web Builder 2\Projects\formsubmit folder. It illustrates the HTML form submittal functionality discussed in the Using HTML Forms topic in this manual.

Controls Layout Example



After installation, the layout.wbp example project will be located in the My Documents\Elevate Web Builder 2\Projects\layout subdirectory. It illustrates how to use the control Layout Management to affect control positioning and sizing.

Responsive Layout Example



Responsive Layout Example

Layout Order: 0
Position: ptTopLeft
Stretch: lsNone
Consumption: lcBottom
Reset: True
Overflow: loNone

Layout Order: 2
Position: ptTopLeft
Stretch: lsRight
Consumption: lcBottom
Reset: True
Overflow: loBottom

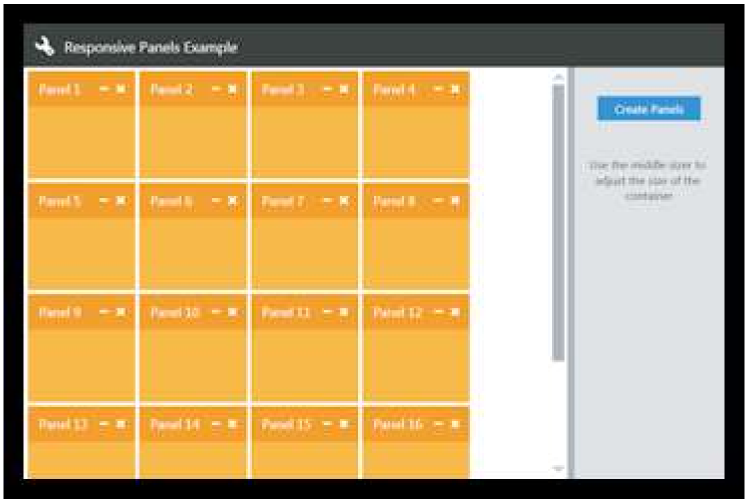
Layout Order: 1
Position: ptTopLeft
Stretch: lsNone
Consumption: lcRight
Reset: True
Overflow: loNone

Layout Order: 3
Position: ptTopLeft
Stretch: lsNone
Consumption: lcRight
Reset: True
Overflow: loBottom

Layout Order: 4
Position: ptTopLeft
Stretch: lsNone
Consumption: lcBottom
Reset: True
Overflow: loBottom

After installation, the responsive.wbp example project will be located in the My Documents\Elevate Web Builder 2\Projects\responsive subdirectory. It illustrates how to use the control Layout Management to build a responsive application that automatically adjusts its interface as the size of the browser window changes.

Responsive Panels Example

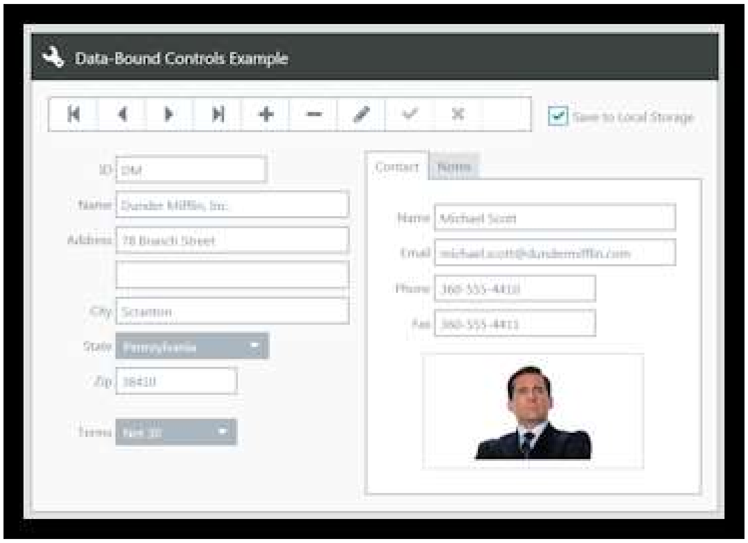


Responsive Panels Example

Use the middle sizer to adjust the size of the container

After installation, the panels.wbp example project will be located in the My Documents\Elevate Web Builder 2\Projects\panels subdirectory. It illustrates how to use the control Layout Management to create a flow layout for panels in a scrollable container.

Data-Bound Controls Example



Data-Bound Controls Example

☑ Save to Local Storage

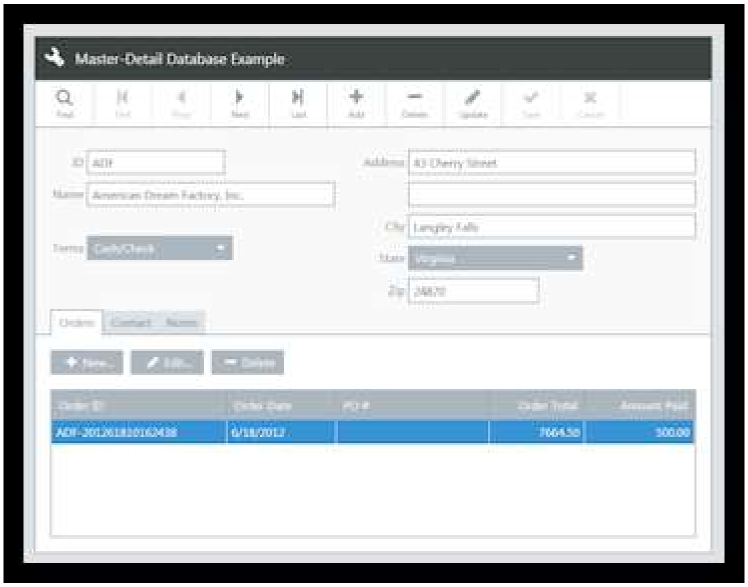
ID: DM
Name: Dunder Mifflin, Inc.
Address: 78 Branch Street
City: Scranton
State: Pennsylvania
Zip: 18410
Terms: Net 30

Contact | Notes

Name: Michael Scott
Email: michael.scott@dundermifflin.com
Phone: 360-555-4410
Fax: 360-555-4411

After installation, the databound.wbp example project will be located in the My Documents\Elevate Web Builder 2\Projects\databound subdirectory. It illustrates the database functionality discussed in the Creating and Loading DataSets, Navigating DataSets, Searching and Sorting DataSets, Updating DataSets, Responding to DataSet Changes, and Binding Controls to DataSets topics in this manual. This project uses the "ExampleData" database that is automatically added to the Database Manager when the example applications are installed.

Master-Detail Database Example



Master-Detail Database Example

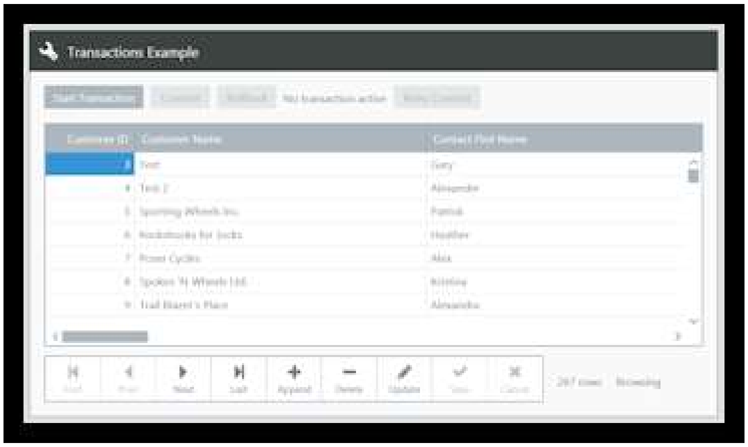
ID: ADF
Name: American Dream Factory, Inc.
Terms: Cash/Check
Address: 43 Cherry Street
City: Langley Falls
State: Virginia
Zip: 24870

Orders | Contact | Notes

Order ID	Order Date	PO #	Order Total	Amount Paid
ADF-201261810162438	6/18/2012		7664.50	500.00

After installation, the masterdetail.wbp example project will be located in the My Documents\Elevate Web Builder 2\Projects\masterdetail subdirectory. It illustrates the database functionality discussed in the Creating and Using Databases, Creating and Loading DataSets, Navigating DataSets, Searching and Sorting DataSets, Updating DataSets, Responding to DataSet Changes, and Binding Controls to DataSets topics in this manual. This project uses the "ExampleData" database that is automatically added to the Database Manager when the example applications are installed.

Transactions Example



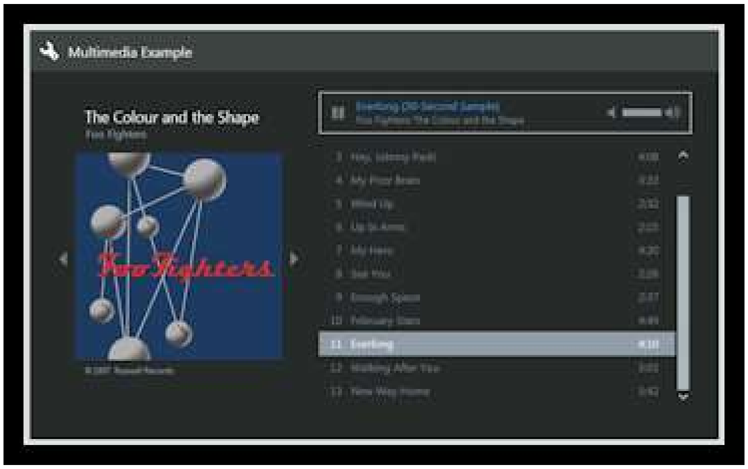
No transaction active

Customer ID	Customer Name	Contact First Name
3	Test	Gary
4	Test 2	Alexandre
5	Sporting Wheels Inc.	Patrick
6	Rockshocks for Jocks	Heather
7	Rosen Cycles	Alex
8	Spokes 'N Wheels Ltd.	Kristina
9	Trail Blazer's Place	Alexandra

267 rows Browsing

After installation, the transactions.wbp example project will be located in the My Documents\Elevate Web Builder 2\Projects\transactions subdirectory. It illustrates the dataset and database transaction functionality discussed in the Creating and Loading DataSets, Navigating DataSets, Searching and Sorting DataSets, Updating DataSets, Responding to DataSet Changes, Binding Controls to DataSets, and Transactions topics in this manual. This project uses the "ExampleDatabase" database that is automatically added to the Database Manager when the example applications are installed.

Multimedia Example



The Colour and the Shape
Foo Fighters
©1997 Roswell Records

Everlong (30-Second Sample)
Foo Fighters The Colour and the Shape

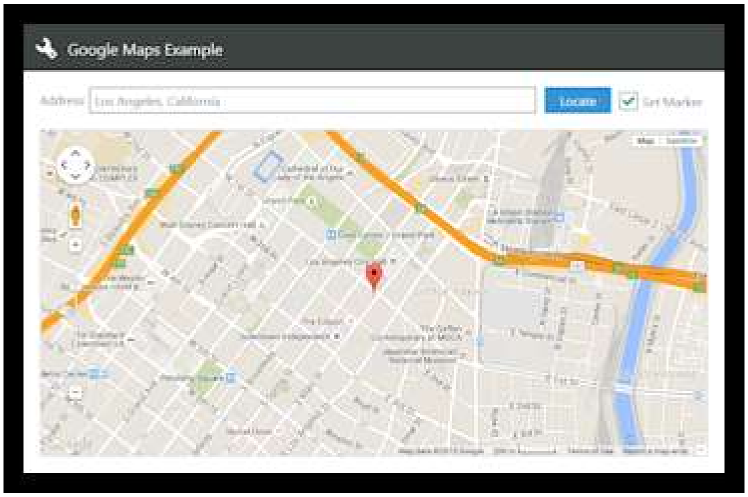
3	Hey, Johnny Park!	4:08
4	My Poor Brain	3:33
5	Wind Up	2:32
6	Up In Arms	2:15
7	My Hero	4:20
8	See You	2:26
9	Enough Space	2:37
10	February Stars	4:49
11	Everlong	4:10
12	Walking After You	5:03
13	New Way Home	5:42

After installation, the multimedia.wbp example project will be located in the My Documents\Elevate Web Builder 2\Projects\multimedia subdirectory. It illustrates the dataset functionality as well as how to use the TAudio control to play audio files. This example project uses the "ExampleDatabase" database that is automatically added to the Database Manager when the example applications are installed.

This example application includes the relevant database tables, but the audio track BLOB fields are empty due to their size and the number of tracks. If you wish to see this example application live, you can do so here:

Elevate Web Builder 2 Multimedia Example

Google Maps Example

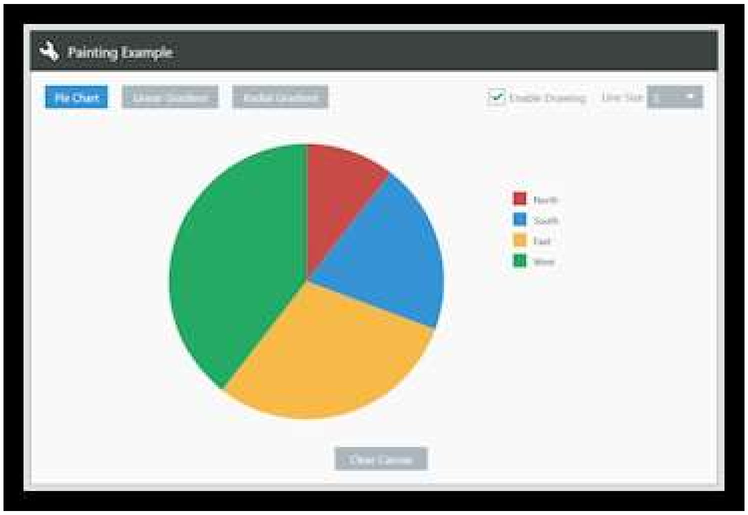


Address: Los Angeles, California

☑ Set Marker

After installation, the maps.wbp example project will be located in the My Documents\Elevate Web Builder 2\Projects\maps subdirectory. It illustrates how to use the TMap control to perform geocoding and mapping using the Google Maps API, as well as how to use the TLocationServices component to obtain the current location from the browser.

Painting Example



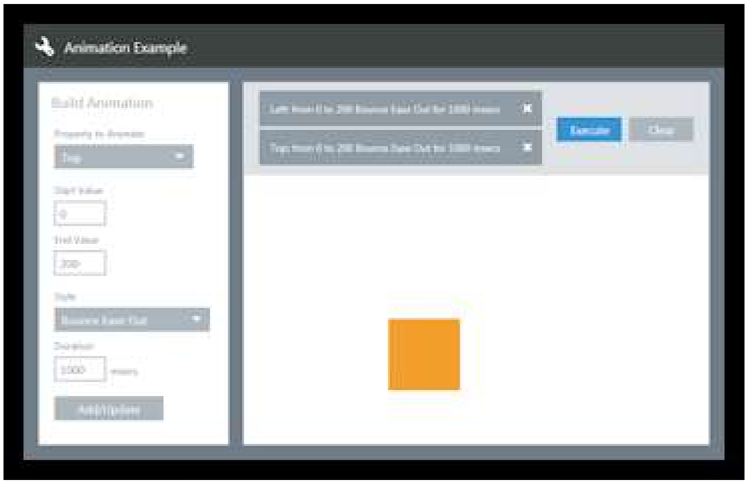
☑ Enable Drawing

After installation, the paint.wbp example project will be located in the My Documents\Elevate Web Builder 2\Projects\paint subdirectory. It illustrates how to use the TPaint control and its TCanvasElement functionality to perform drawing operations.

Slideshow Example

After installation, the slideshow.wbp example project will be located in the My Documents\Elevate Web Builder 2\Projects\slideshow subdirectory. It illustrates how to use the TSlideshow control to show a slideshow of images. This example project uses the "ExampleData" database that is automatically added to the Database Manager when the example applications are installed.

Animation Example



Property to Animate: Top

Start Value: 0

End Value: 200

Style: Bounce Ease Out

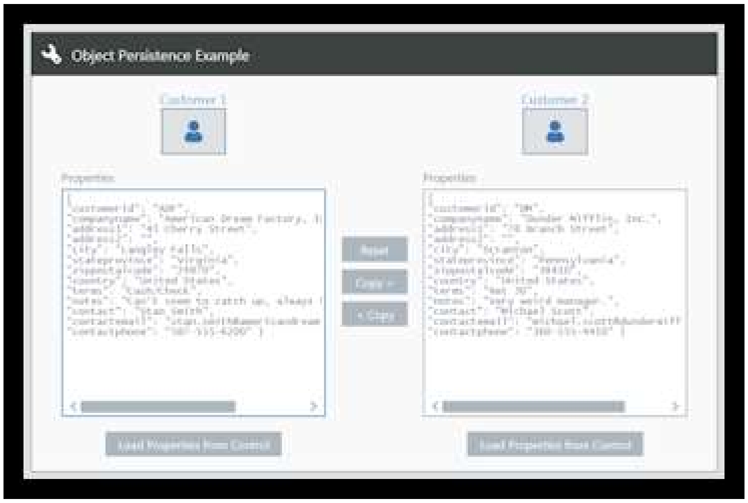
Duration: 1000 msecs

Left from 0 to 200 Bounce Ease Out for 1000 msecs

Top from 0 to 200 Bounce Ease Out for 1000 msecs

After installation, the animation.wbp example project will be located in the My Documents\Elevate Web Builder 2\Projects\animation subdirectory. It illustrates how to use the Animations properties of controls to perform animation operations.

Object Persistence Example

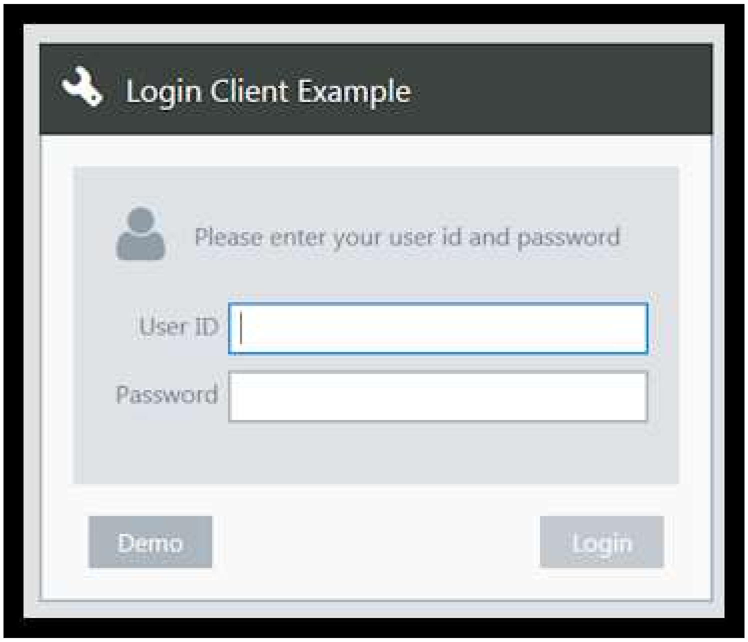

Object Persistence Example
Customer 1
Customer 2
Properties
Properties
Reset
Copy >
< Copy
Load Properties from Control
Load Properties from Control

```
{
"customerid": "ADF",
"companyname": "American Dream Factory, I
"address1": "41 Cherry Street",
"address2": "",
"city": "Langley Falls",
"stateprovince": "Virginia",
"zippostalcode": "39870",
"country": "United States",
"terms": "Cash/Check",
"notes": "Can't seem to catch up, always !
"contact": "Stan Smith",
"contactemail": "stan.smith@americandream
"contactphone": "587-555-4200" }
```

```
{
"customerid": "DM",
"companyname": "Dunder Mifflin, Inc.",
"address1": "78 Branch Street",
"address2": "",
"city": "Scranton",
"stateprovince": "Pennsylvania",
"zippostalcode": "39432",
"country": "United States",
"terms": "Net 30",
"notes": "Very weird manager.",
"contact": "Michael Scott",
"contactemail": "michael.scott@dundermiff
"contactphone": "360-555-9410" }
```

After installation, the persistence.wbp example project will be located in the My Documents\Elevate Web Builder 2\Projects\persistence subdirectory. It illustrates how to use the persistence functionality to load/save published properties to/from TPersistent descendant classes.

Login Client Example



Please enter your user id and password

User ID

Password

After installation, the loginclient.wbp example project will be located in the My Documents\Elevate Web Builder 2\Projects\loginclient subdirectory. It illustrates how to authenticate a user ID and password using the TServerRequest component and a web server module project (see next).

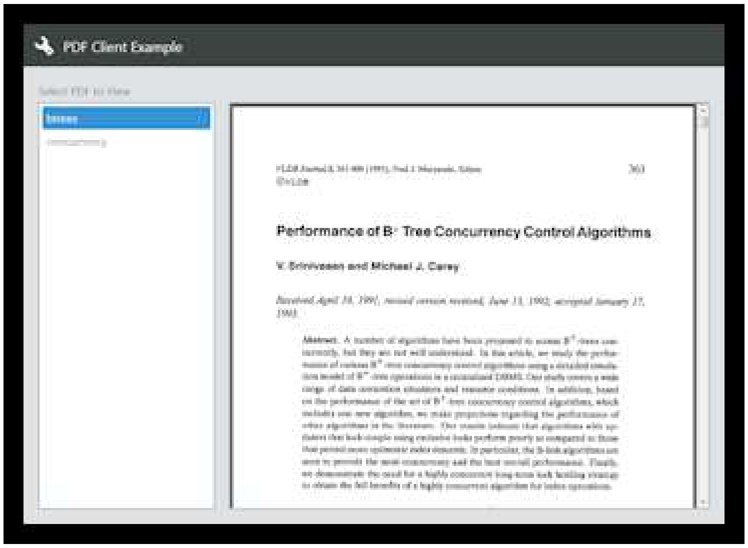
Login Module Example

After installation, the loginmodule.dpr example Delphi project will be located in the My Documents\Elevate Web Builder 2\Projects\loginmodule subdirectory. Unlike the other client projects mentioned above, the loginmodule.dpr example project is a server-side Delphi project that shows how to create a web server module for authenticating a login using a user ID and password. Please see the Creating Web Server Modules topic for more information on downloading the Elevate Web Builder 2

Modules installation for your version of Embarcadero RAD Studio and Delphi. This download must be installed before you can begin creating web server modules for use with Elevate Web Builder applications.

In addition, a pre-compiled copy (loginmodule.dll) of the loginmodule example project will be located in the \bin\loginmodule\win32 subdirectory under the main installation directory, and this pre-compiled web server module will be added to the IDE during the example installation so that it can be used with the Login Client example project above.

PDF Client Example


PDF Client Example
363
Performance of B⁺ Tree Concurrency Control Algorithms
V. Srinivasan and Michael J. Carey

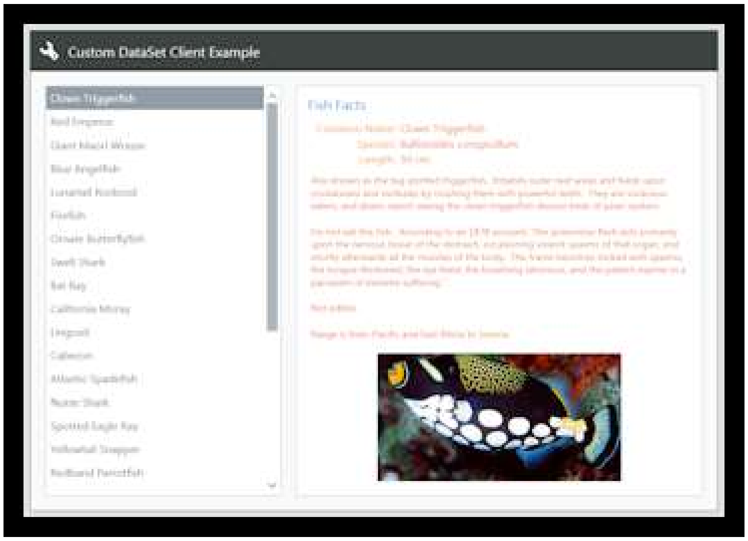
After installation, the pdfclient.wbp example project will be located in the My Documents\Elevate Web Builder 2\Projects\pdfclient subdirectory. It illustrates how to dynamically request and load PDF files from the Elevate Web Builder Web Server using the TServerRequest component, the TPlugin control, and a web server module project (see next).

PDF Module Example

After installation, the pdfmodule.dpr example Delphi project will be located in the My Documents\Elevate Web Builder 2\Projects\pdfmodule subdirectory. Unlike the other client projects mentioned above, the pdfmodule.dpr example project is a server-side Delphi project that shows how to create a web server module for loading PDF files from a server directory. Please see the Creating Web Server Modules topic for more information on downloading the Elevate Web Builder 2 Modules installation for your version of Embarcadero RAD Studio and Delphi. This download must be installed before you can begin creating web server modules for use with Elevate Web Builder applications.

In addition, a pre-compiled copy (pdfmodule.dll) of the pdfmodule example project will be located in the \bin\pdfmodule\win32 subdirectory under the main installation directory, and this pre-compiled web server module will be added to the IDE during the example installation so that it can be used with the PDF Client example project above.

Database Module Client Example



Custom DataSet Client Example

Clown Triggerfish
Red Emperor
Giant Maori Wrasse
Blue Angelfish
Lunartail Rockcod
Firefish
Ornate Butterflyfish
Swell Shark
Bat Ray
California Moray
Lingcod
Cabezon
Atlantic Spadefish
Nurse Shark
Spotted Eagle Ray
Yellowtail Snapper
Redband Parrotfish

Fish Facts

Common Name: Clown Triggerfish
Species: Balistoides conspicillum
Length: 50 cm

[illegible]

[illegible]

[illegible]

[illegible]

After installation, the databaseclient.wbp example project will be located in the My Documents\Elevate Web Builder 2\Projects\databaseclient subdirectory. It illustrates how to load datasets from the Elevate Web Builder Web Server using a database module project (see next).

Database Module Example

After installation, the databasemodule.dpr example Delphi project will be located in the My Documents\Elevate Web Builder 2\Projects\databasemodule subdirectory. Unlike the other client projects mentioned above, the databasemodule.dpr example project is a server-side Delphi project that shows how to create a database module for loading a dataset from an ElevateDB database and demonstrates the usage of the TEWBDatabaseAdapter and TEWBDataSetAdapter components for generating/consuming JSON from TDataSet-descendant component instances in Embarcadero RAD Studio and Delphi. Please see the Creating Web Server Modules topic for more information on downloading the Elevate Web Builder 2 Modules installation for your version of Embarcadero RAD Studio and Delphi. This download must be installed before you can begin creating web server modules for use with Elevate Web Builder applications.

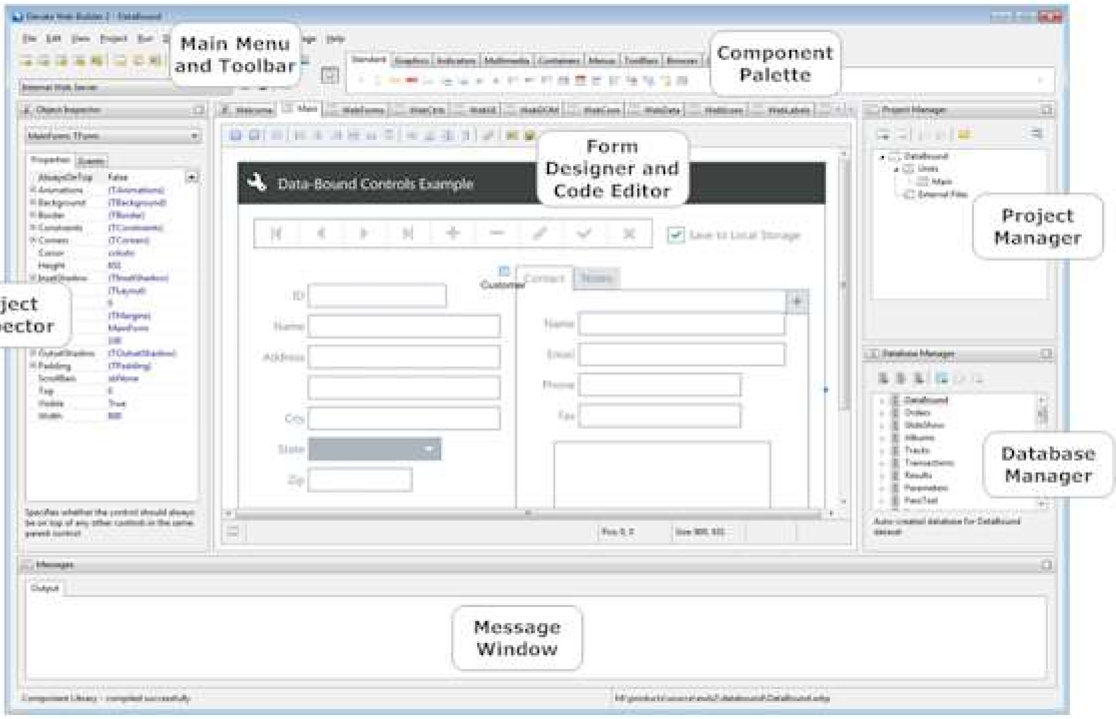
In addition, a pre-compiled copy (databasemodule.dll) of the databasemodule example project will be located in the \bin\databasemodule\win32 subdirectory under the main installation directory, and this pre-compiled database module will be added to the IDE during the example installation so that it can be used with the Database Module Client example project above.

This page intentionally left blank

Chapter 2
Using the IDE

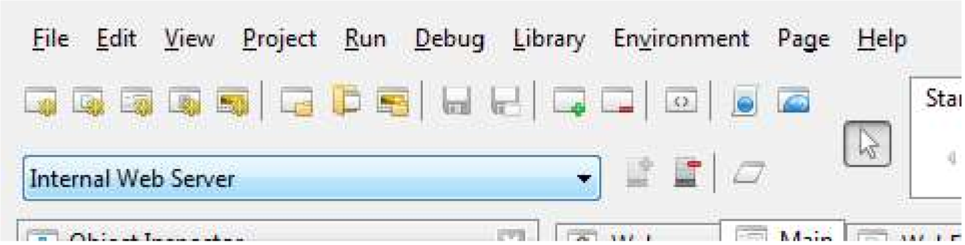
2.1 Introduction

The Elevate Web Builder IDE is comprised of several distinct parts, each with their own specific functionality as it relates to the development process. The various parts of the IDE are illustrated below:



The main menu and toolbar provide options for:

- Creating new visual or non-visual projects
- Creating new forms, databases, and source units in a project
- Adding existing forms, databases, and source units to an existing project
- Modifying project options
- Compiling, deploying, and running projects
- Saving and closing projects
- Viewing units and forms
- Adding components to, removing components from, and rebuilding the component library
- Creating new control interfaces and modifying existing control interfaces



Note
The Debug option on the main menu is not enabled at this time and is not functional. It will be used to enable debugging of design-time code at a later time.

The middle page control of the IDE is reserved for all open source units, forms, and databases. Each source unit, form, and database is docked to a page in the page control and presented using the form or database designer and code editor.

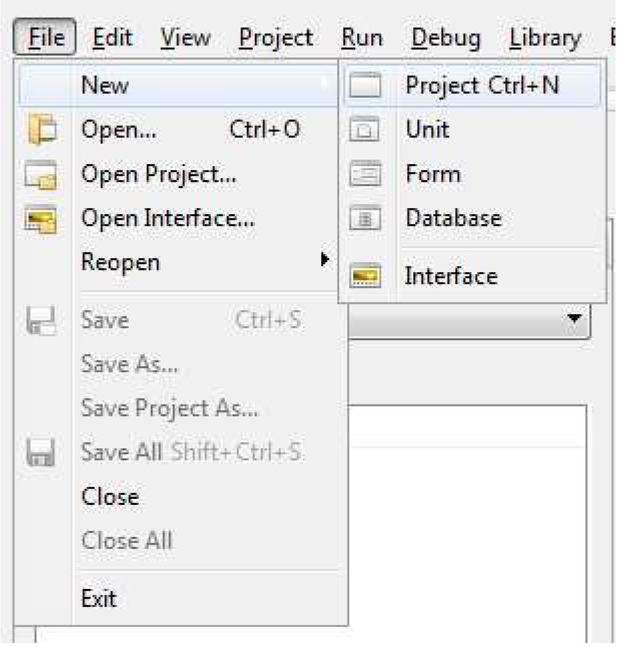
Note
Source units that aren't associated with a form or database only use the code editor, and not the form or database designer.

Control interfaces are also docked to the middle page control when opened, and are presented using the interface editor. Control interfaces are not associated with a specific project, but open control interfaces are saved with open projects so that they are re-opened whenever the project is re-opened.

2.2 Creating a New Project

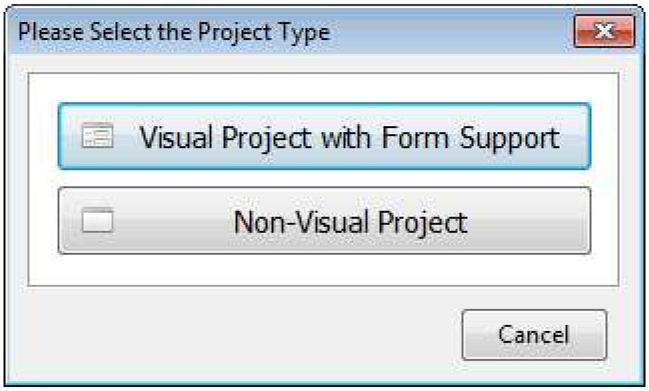
Use the following steps to create a new application project in the IDE:

- Click on the **File** option in the main menu. The File menu will open:



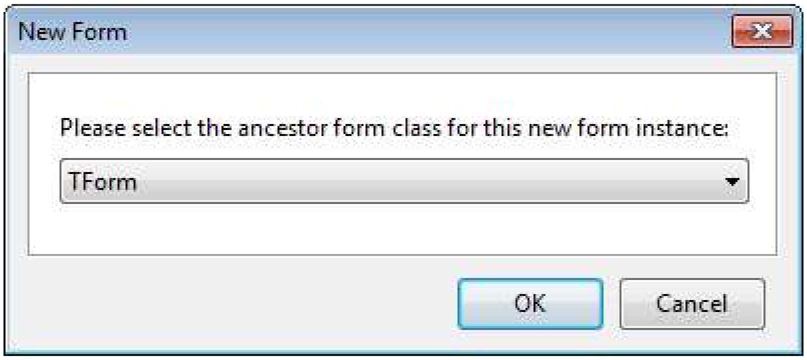
- Click on the **New** option in the File menu to open the New sub-menu. From the New sub-menu, select the **Project** option.

- A dialog will be displayed that will allow you select the type of project that you would like to create, either visual or non-visual:


Please Select the Project Type
Visual Project with Form Support
Non-Visual Project
Cancel

Select the desired project type, and the new project will be opened in the IDE.

- If you selected a visual project type, you will then be prompted to select the type of form class to use as the ancestor of the main form for the visual project:



New Form

Please select the ancestor form class for this new form instance:

TForm

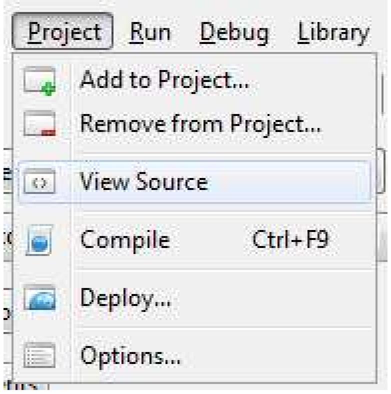
If you're unsure as to which form class to use, just use the default TForm class.

Visual Projects vs. Non-Visual Projects

The main difference between visual projects and non-visual projects is the way that the IDE generates code in the project source file as forms/source units are added and removed from the project.

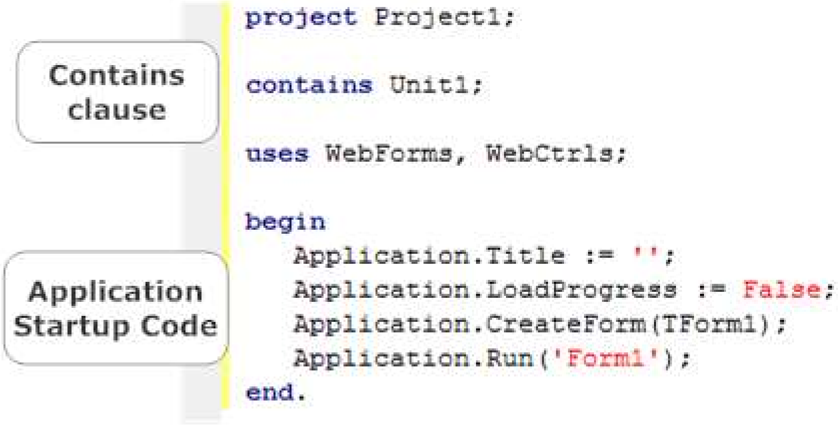
Use the following steps to access the project source file:

- Click on the **Project** option in the main menu. The Project menu will open:



- Click on the **View Source** option in the Project menu to open the project source file in the code editor.

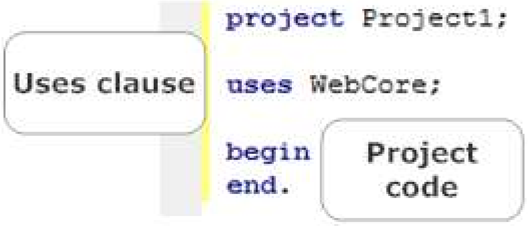
For visual projects, the project source file's contains clause is updated to include the name of the source units included in the project, as well as the application startup code for the TApplication component that is used with visual applications. In addition, any forms that are marked as auto-create forms in the Project Options are automatically created here. You can see how this looks in the following image:



```
project Project1;

contains Unit1;

uses WebForms, WebCtrls;

begin
   Application.Title := '';
   Application.LoadProgress := False;
   Application.CreateForm(TForm1);
   Application.Run('Form1');
end.
```

Note
You'll notice that the IDE automatically inserts the WebForms, and WebCtrls units into the uses clause. These units are necessary to support the generated application startup code, and should never be removed.

For non-visual projects, the project source file's uses clause is updated to include the name of the source units included in the project only. You can see how this looks in the following image:



```
project Project1;

uses WebCore;

begin
end.
```

The user code would be added in the begin..end block.

Note
You'll notice that the IDE automatically inserts the WebCore unit into the uses clause. This unit is necessary, and should never be removed.

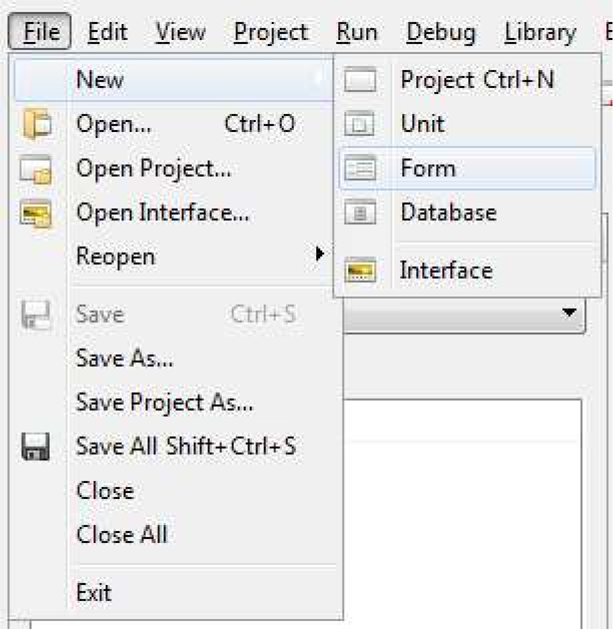
2.3 Adding to an Existing Project

You can easily add new forms, databases, and units to an existing project in Elevate Web Builder.

Adding a Form to a Project

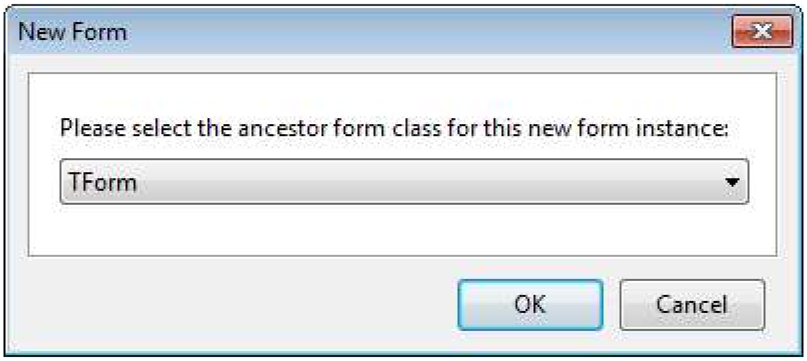
Use the following steps to add a new form to an existing project:

- Click on the **File** option in the main menu. The File menu will open:



- Click on the **New** option in the File menu to open the New sub-menu. From the New sub-menu, select the **Form** option.

- You will then be prompted to select the type of form class to use as the ancestor of the form for the visual project:



Please select the ancestor form class for this new form instance:

TForm

If you're unsure as to which form class to use, just use the default TForm class.

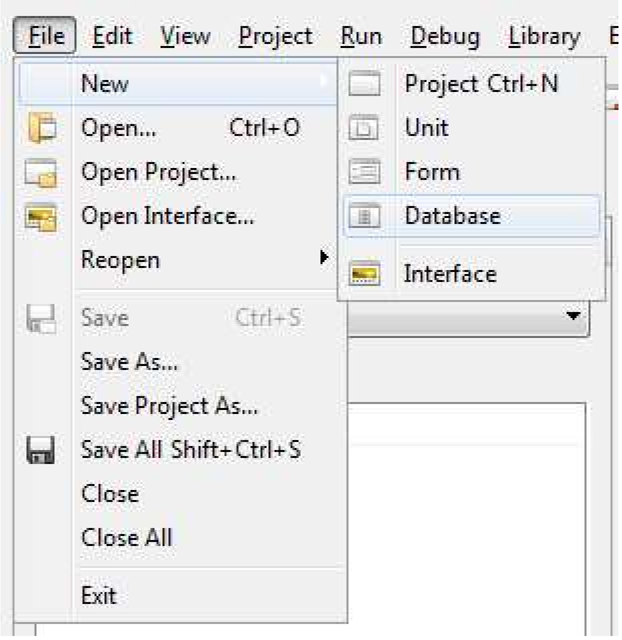
- A new form will now appear in the form designer. Please see Using the Form and Database Designers. for more information on how to use the form designer.

Note
The Form option is only available from the New sub-menu when a visual project is active. You cannot add new forms to non-visual projects.

Adding a Database to a Project

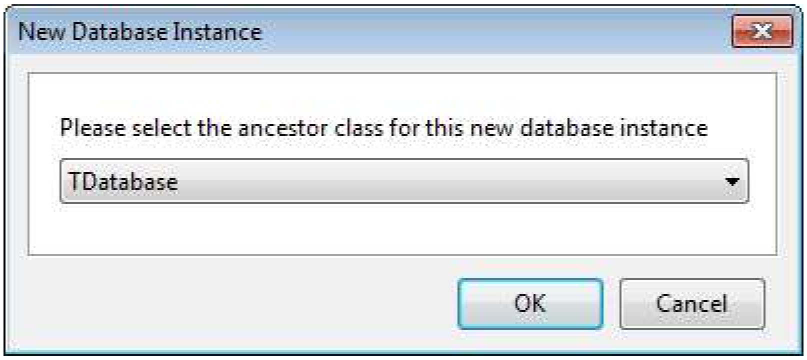
Use the following steps to add a new database to an existing project:

- Click on the **File** option in the main menu. The File menu will open:



- Click on the **New** option in the File menu to open the New sub-menu. From the New sub-menu, select the **Database** option.

- You will then be prompted to select the type of database class to use as the ancestor of the database for the visual project:


New Database Instance

Please select the ancestor class for this new database instance

TDatabase

If you're unsure as to which database class to use, just use the default TDatabase class.

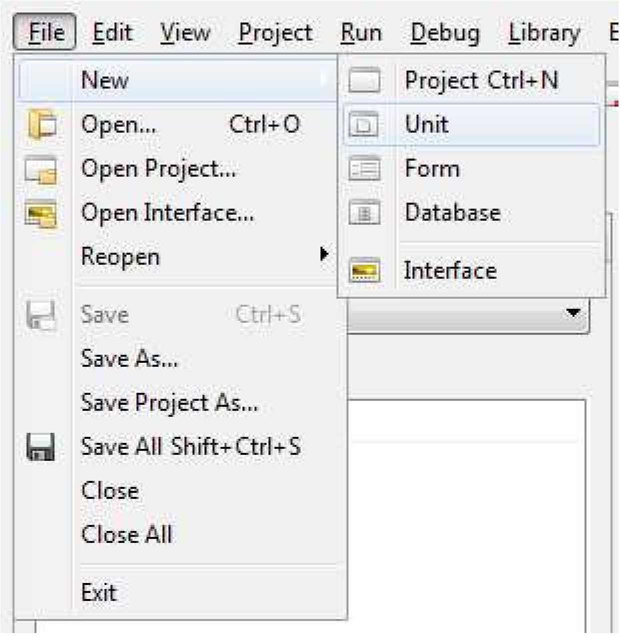
- A new database will now appear in the database designer. Please see Using the Form and Database Designers. for more information on how to use the database designer.

Note
The Database option is only available from the New sub-menu when a visual project is active. You cannot add new databases to non-visual projects.

Adding a Source Unit to a Project

Use the following steps to add a new source unit to an existing project:

- Click on the **File** option in the main menu. The File menu will open:



- Click on the **New** option in the File menu to open the New sub-menu. From the New sub-menu, select the **Unit** option.

- A new source unit will now appear in the code editor. Please see the Using the Code Editor topic for more information on using the code editor.

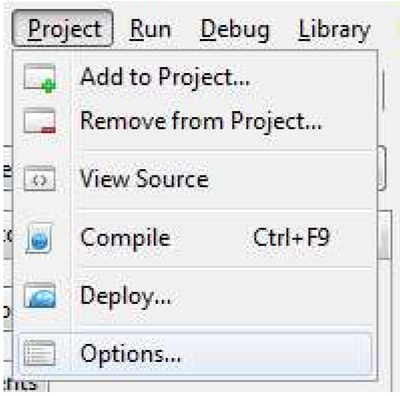
2.4 Modifying Project Options

The project options for a project include:

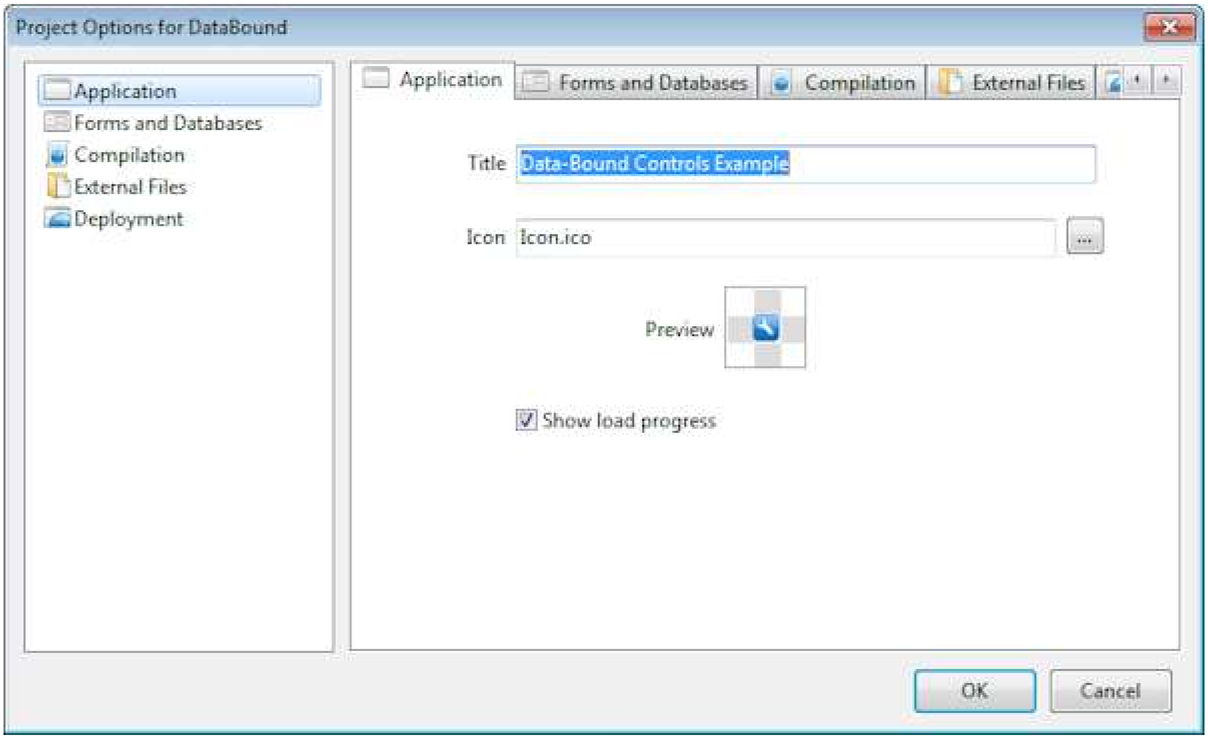
- General application options (title, icon, whether to show load progress)
- Auto-creation of forms and databases in visual applications
- Compilation options (search paths, output paths, output compression)
- External Files
- Deployment options

Use the following steps to modify the project options for a project:

- Click on the **Project** option in the main menu. The Project menu will open:



- Click on the **Options** option in the Project menu to open the Project Options dialog.



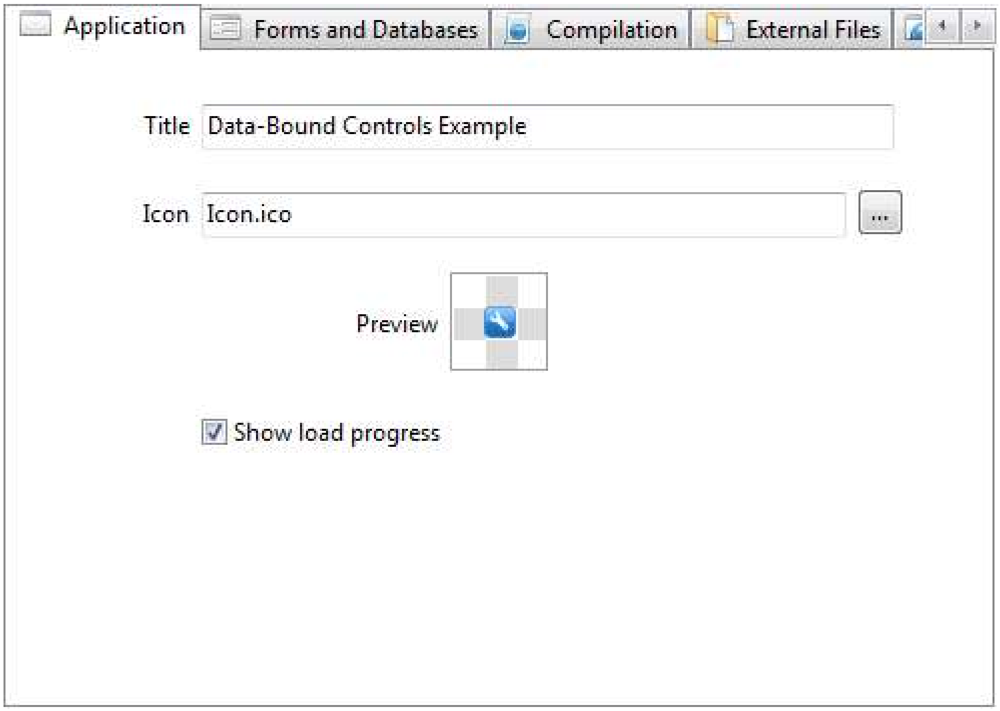
Title Data-Bound Controls Example

Icon Icon.ico ...

☑ Show load progress

Application

For visual applications, the Application page provides options for specifying the title of the application, the icon to display in the browser window for the application, and whether or not to show load progress.



Title Data-Bound Controls Example

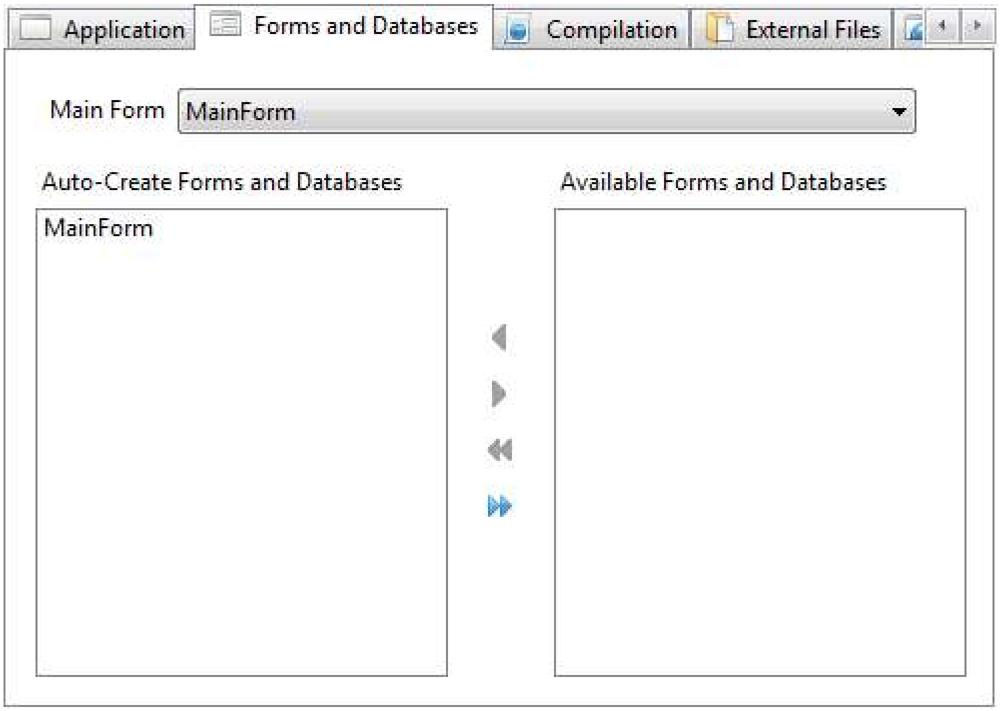
Icon Icon.ico

☑ Show load progress

Option	Description
Title	The application title is the descriptive name for the application and, in most modern browsers, will appear in the caption bar of the browser window.
Icon	The application icon is a 16x16 or 32x32 Windows icon file that is displayed in the browser window next to the application title. This icon is commonly known as a "favicon" (short for "favorite icon") because the icon is also used to help identify the application in "favorites" or "bookmarks" in the browser. You can type in the file name directly, or use the browse button (...) to select the icon file using a common Windows file dialog. After a valid file name has been specified or selected, a preview of the icon file will be shown in the Preview area. **Note** You do not have to specify an icon for an application. It is completely optional.
Show load progress	If checked, this option will turn on the load progress dialog for the application. This dialog is shown while all forms marked as auto-create are being created. See the Forms section below for more information on determining which forms will be auto-created.

Forms and Databases

For visual applications, the Forms and Databases page allows you to specify which forms and databases in the project should be auto-created, and which forms and databases should not be auto-created. The first form in the list of auto-create forms and databases is automatically designated as the main form of the application, but you can select a different auto-create form as the main form using the combo box at the top of the page.



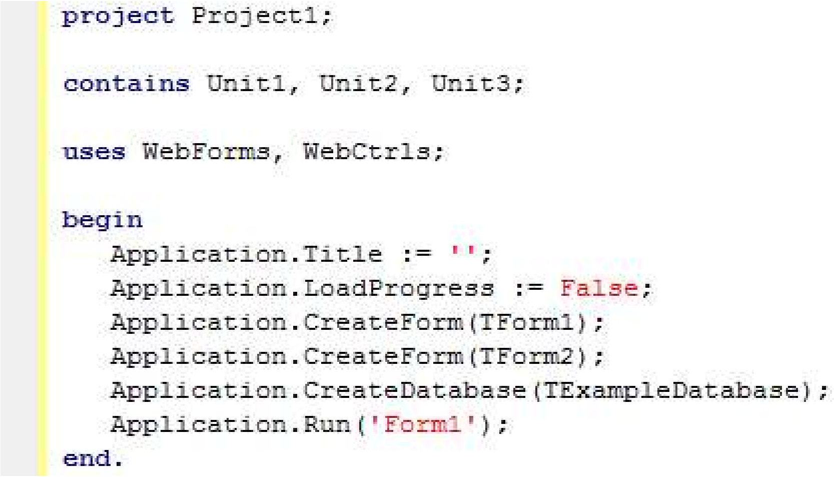
Main Form MainForm

Auto-Create Forms and Databases

MainForm

Available Forms and Databases

All updates to the main form and/or the auto-create forms and databases list will be reflected in the project source file. The following shows the project source file that corresponds to the auto-create forms and databases list above:



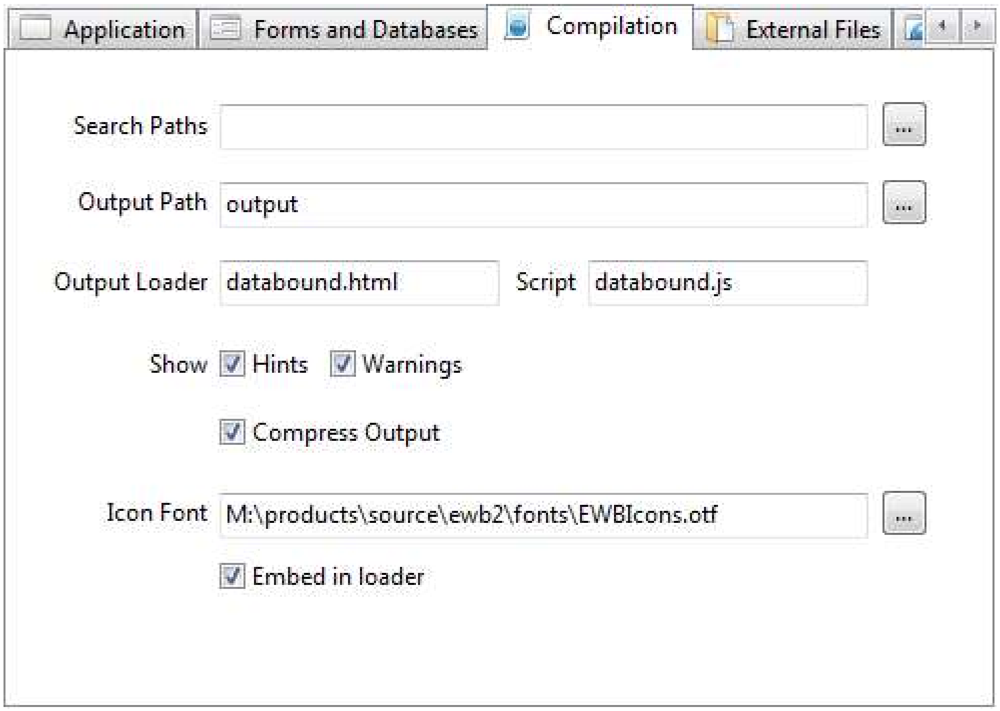
```
project Project1;

contains Unit1, Unit2, Unit3;

uses WebForms, WebCtrls;

begin
   Application.Title := '';
   Application.LoadProgress := False;
   Application.CreateForm(TForm1);
   Application.CreateForm(TForm2);
   Application.CreateDatabase(TExampleDatabase);
   Application.Run('Form1');
end.
```

Option	Description
Main Form	The main form is set to the first form that is designated as auto-create, or blank if no forms are designated as auto-create. You can select a different main form by using this combo box.
Auto-Create Forms and Databases	The IDE can be configured, via the Environment Options dialog, to automatically add any new forms and databases created or added to a project to this list. If you don't want a form or database to be automatically created, you can move the form or database to the available list box by dragging and dropping the desired form or database into the Available Forms list box. You can select multiple forms and databases to drag and drop by holding down the Ctrl key and selecting the forms and databases using the mouse.
Available Forms and Databases	This list box shows all forms and databases that are part of the project, but aren't marked as auto-create.

Warning
If you try to show or hide a form that has not been created yet, and is not set as auto-create, you will get a run-time error in the web browser. Likewise, a similar run-time error will occur if you try to access any components on a form or database that haven't been created yet, such as trying to access a dataset in a database.

Compilation

The Compilation page allows you to configure the compilation options for both visual and non-visual projects.



Search Paths

Output Path: output

Output Loader: databound.html Script: databound.js

Show ☑ Hints ☑ Warnings

☑ Compress Output

Icon Font: M:\products\source\ewb2\fonts\EWBIcons.otf

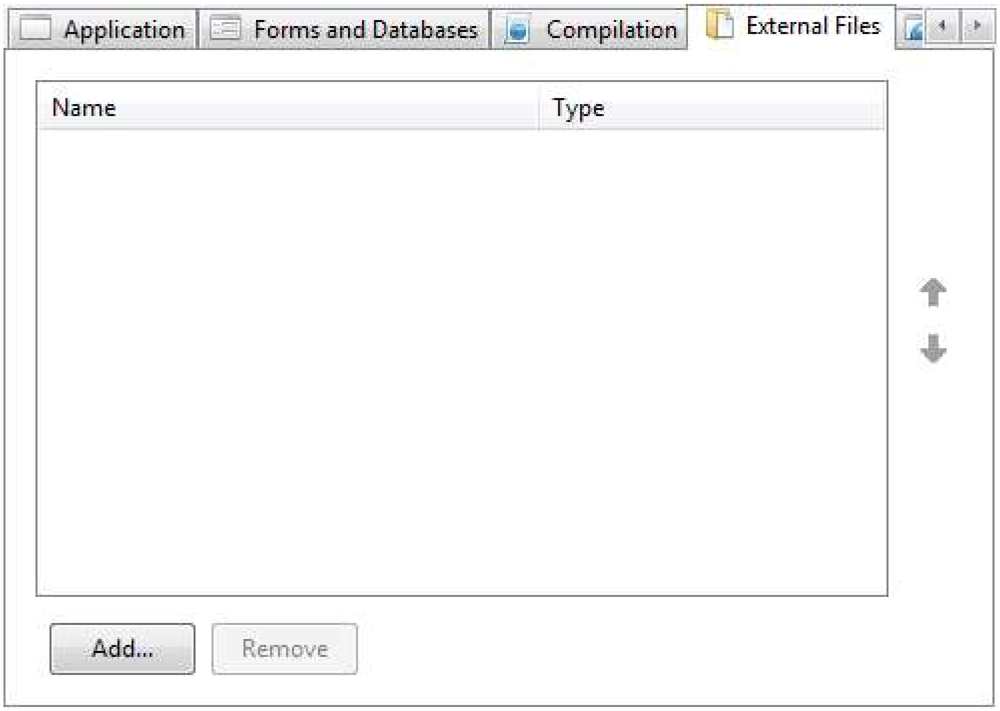
☑ Embed in loader

Option	Description
Search Paths	In many cases you will not need to include any additional compilation search paths for a project. By default, the compiler will look in the project source folder and the component library search paths for any referenced units. Please see the Modifying Environment Options topic for more information on modifying the component library search paths. However, in certain cases you may want to include additional search paths for common library source units or custom control interfaces that are used between multiple projects, and this is where you would do so. When specifying more than one search path, be sure to separate multiple paths with a semicolon (;).
Output Path	This path specifies the output path where the application HTML (.html) loader file and application JavaScript (.js) source file will be emitted. This path is relative to the main project source folder. If you specify an absolute path here, the IDE will automatically convert it to a relative path when the Project Options dialog is closed by clicking on the OK button.
Output Loader	This file name specifies the emitted output name of the application HTML (.html) loader file.
Output Script	This file name specifies the emitted output name of the application JavaScript (.js) file.
Show Hints/Warnings	Make sure these check boxes are selected (default) in order to see all hints and warnings from the compiler about unused variables and other compilation conditions that you may need to know about.

Compressed Output	When this check box is selected, the compiler will emit the HTML and JavaScript for the application in a highly-compressed and obfuscated form. This normally can reduce the size of the resulting HTML and JavaScript files by 50% or more.
Icon Font	This file name specifies the icon font file to use for the embedded icons used with Elevate Web Builder. The icon font file name can use absolute or relative paths, but it is recommended that you use an absolute path in the file name so that there isn't any issue with the compiler finding the icon font file. By default, the icon font file is set to the default icon font file EWBIcons located in the \fonts subdirectory under the main installation directory. Please see the Icon Library topic for more information on using icon fonts with Elevate Web Builder.
Embed in loader	This check box controls whether the specified icon font file name is embedded directly in the HTML loader file created when compiling an application, or whether a link to the icon font file name is used instead. By default, the icon font file will be embedded in the HTML loader file. Please see the Compiling Applications topic for more information on compiling projects with Elevate Web Builder.

External Files

The External Files page allows you to configure which external files (external Javascript, images, etc.) you want to include with your project.



When you include an external file with your project, the compiler will copy the source file to the output path for the project and, if necessary, emit a reference to this source file in the HTML loader file that is also emitted into the output path during compilation. Some external files such as external Javascript source code require a link to the file in the emitted HTML loader file. Please see the External Interfaces topic for more information on interfacing external JavaScript source code in your application source code.

Deployment

The Deployment page allows you to configure how your project should be deployed when the Deploy option is selected from the main menu or main toolbar. There are two deployment methods currently available for a project:

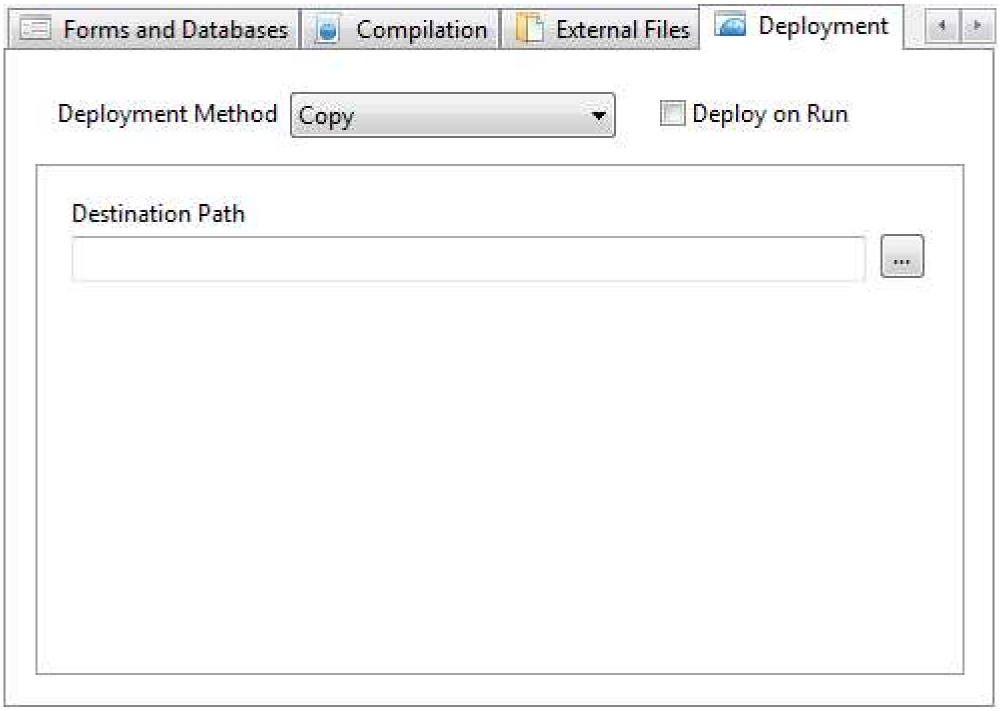
Method	Description
Copy	This is the default method and only requires a destination path name to use for the destination of the copy operation.
FTP	This deployment method will use the File Transfer Protocol (FTP) to copy all output files for the application to the specified destination path on the specified FTP server.

Warning
During deployment, the IDE will try to create any output directories that are required, so you should make sure that you have the proper user privileges for the destination path for either deployment method.

Deploying When an Application is Run

Use the Deploy On Run check box to select whether the application should automatically deployed before it is run in the IDE. This is useful for applications that are being run from an external web server and need to be deployed to the external web server prior to being run. This option is ignored when an application is run from the internal web server that is embedded in the IDE.

Copy



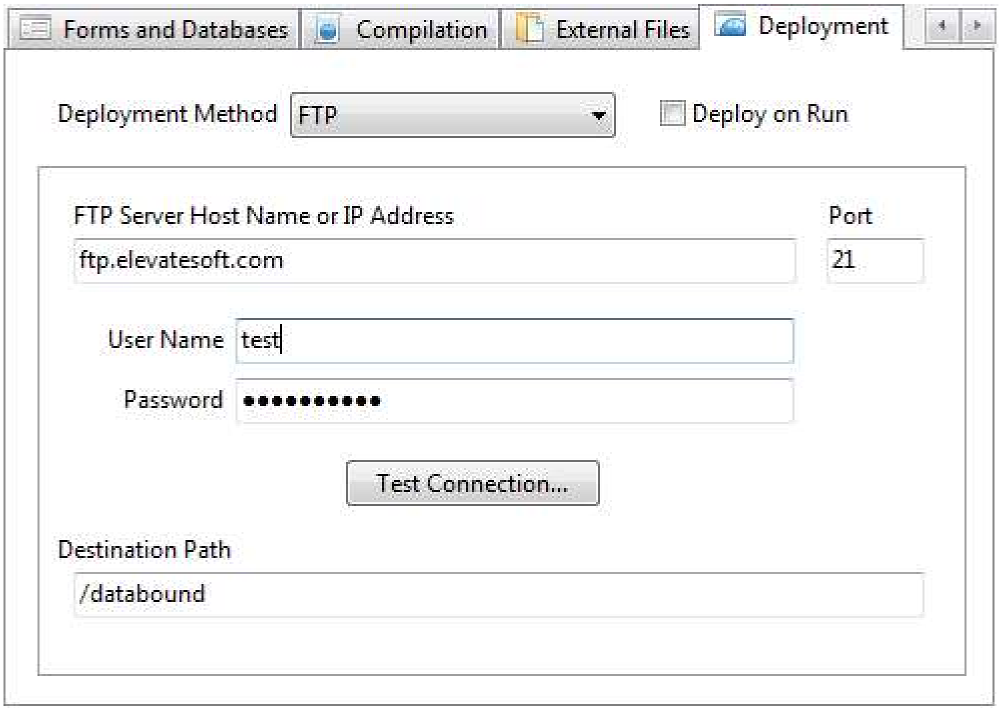
Deployment Method: Copy

□ Deploy on Run

Destination Path:

Option	Description
Destination Path	This is the path where all application output files will be copied. The default value is blank (""), and you must specify a path or the deployment will fail with an error.

FTP



Deployment Method: FTP

□ Deploy on Run

FTP Server Host Name or IP Address: ftp.elevatesoft.com

Port: 21

User Name: test

Password: ●●●●●●●●●●●

Destination Path: /databound

Option	Description
FTP Server Host Name or IP Address	This option specifies the host name (domain name) or IP address (XXX.XXX.XXX) of the FTP server where the application output files should be deployed.
Port	This is the port number on which the FTP server is listening. The default port for FTP servers is port 21.
User Name	This is the user name to use when logging in to the FTP server. If the FTP server does not require a user name and password, then leave this option blank (the default).
Password	This is the password to use when logging in to the FTP server. If the FTP server does not require a user name and password, then leave this option blank (the default).
Destination Path	This is the path where all application output files will be copied. The default value is blank (""), and this indicates to copy all application output files to the root directory of the FTP server.

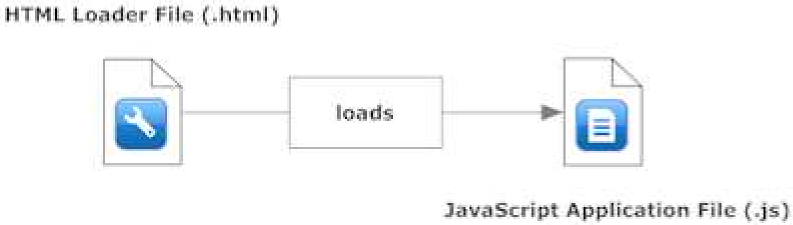
The Test Connection button can be used to verify that you configured the FTP server and login options correctly. If you have done so properly, then you will see a message dialog affirming the fact that the IDE was able to successfully connect and login to the specified FTP server. If there is an error making the connection or logging in with the specified user name and password, then you will see a message dialog with the appropriate error message that indicates the problem.

2.5 Compiling a Project

When a project is compiled, Elevate Web Builder performs the following steps:

- The project source file is compiled.
- All source units referenced in the project source file are compiled. In each source unit, all referenced source units are compiled, and this continues until all referenced source units are compiled.
- After all referenced source units are compiled, the application is emitted. During the emitting phase, the compiler creates a single loader HTML (Hyper-Text Markup Language) file with the same root name as the project, and a single JS (JavaScript) file with the same root name as the project.

The following illustrates the compiled output of an Elevate Web Builder application:

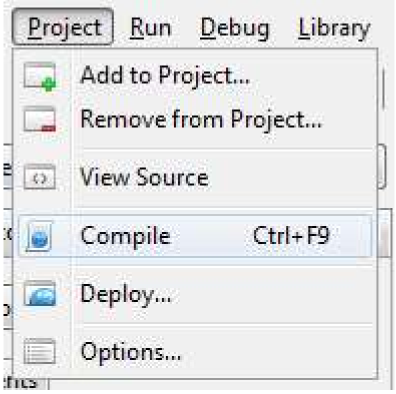


Elevate Web Builder Runtime Application Structure

By default, all output files are emitted in an "output" folder located within the same folder as the project. Please see the Modifying Project Options topic for more information on modifying the compilation output folders.

Use the following steps to compile a project:

- Click on the **Project** option in the main menu. The Project menu will open:



- Click on the **Compile** option in the Project menu to compile the current project. If there are any hints, warnings, or errors during compilation, they will appear in the Messages panel at the bottom of the IDE. If any errors are present, the compilation will fail and the application output files will not be emitted.

You can also use the keyboard to compile an application by holding down the Ctrl key and hitting the F9 key.

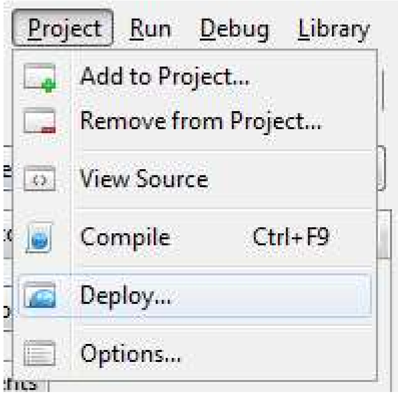
2.6 Deploying a Project

A project can be deployed using a straight copy method (default), or by using a connection to an FTP server. Please see the Modifying Project Options topic for more information on selecting the deployment method, and to generally configure the deployment.

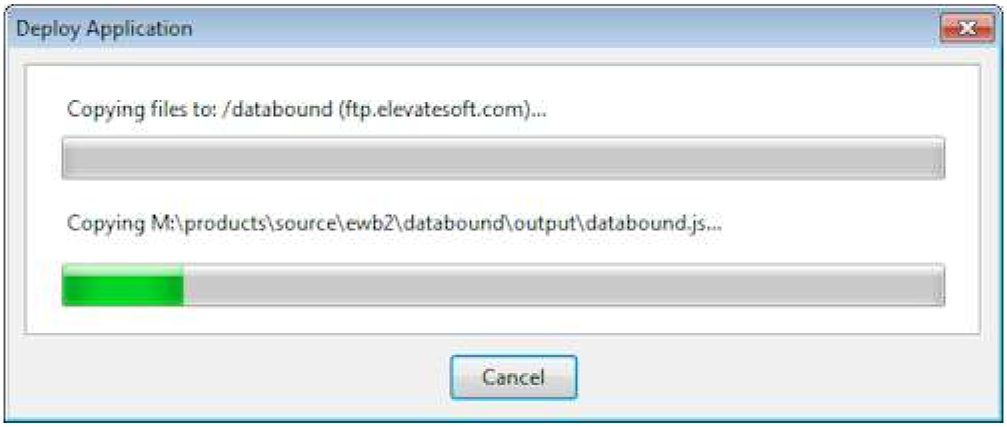
> **Note**
> The last compiled version of a project is what will be copied to the destination path when a project is deployed. It is always wise to make sure to compile a project before deploying in order to ensure that the most recent version of the application is properly copied.

Use the following steps to deploy a project:

- Click on the **Project** option in the main menu. The Project menu will open:



- Click on the **Deploy** option in the Project menu to deploy the current project. During deployment, information about each application file being copied will appear in the Messages panel at the bottom of the IDE. In addition, a progress dialog will be displayed that shows the total progress of the application deployment, as well as the progress of the current file being copied:



Copying files to: /databound (ftp.elevatesoft.com)...

Copying M:\products\source\ewb2\databound\output\databound.js...

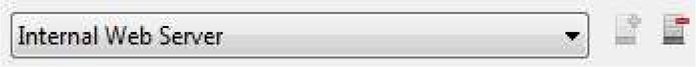
2.7 Running a Project

It is possible to run projects directly in the IDE for testing purposes. The IDE uses an embedded version of the Internet Explorer web browser to run the application.

> **Note**
> Elevate Web Builder requires that Internet Explorer 9 or higher be installed in order to properly run applications in the IDE.

Use the following steps to run a project:

- Use the web server combo box to select the web server that you want to run the project from:

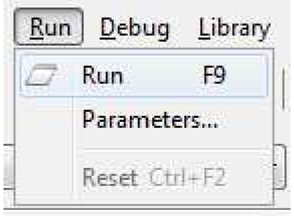


By default, the internal web server embedded in the IDE is automatically set as the default web server. You can add external web servers by using the **External Web Servers** tab in the Environment Options dialog.

To the right of the web server combo box are two buttons that can be used to start and stop the selected web server. These options only work with the internal web server embedded in the IDE and are unavailable for any external web servers.

> **Note**
> If the internal web server is selected, but is not started, then the local file system will be used to run the application. We recommend that you not run the application from the local file system unless the application doesn't contain any databases and doesn't execute any server requests. Attempting to run an application that uses these features from the local file system will result in numerous runtime errors.

- Click on the **Run** option in the main menu. The Run menu will open:



Click on the **Run** option in the Run menu to run the current project. The IDE will automatically compile the application before running it. If there are any hints, warnings, or errors during compilation, they will appear in the Messages panel at the bottom of the IDE. If any errors are present, the compilation will fail, the application output files will not be emitted, and the application will not run.

If deployment has been configured for the application via the **Deployment** tab in the Project Options dialog and the Deploy On Run option has been selected, then the application will be automatically deployed after it has been successfully compiled and before it is actually run in the web browser in the IDE. If there are any errors during deployment, or if the deployment is cancelled, then the application will not run. The Deploy On Run option is ignored when the selected web server is the internal web server.

> **Note**
> If you are running the application from an external web server, then it is very important that you configure the deployment for the application, being sure to select the Deploy On Run option and then ensure that the deployment settings are accurate for the external web server. Failure to do so will result in an outdated version of the application running from the external web server.

You can also use the keyboard to run an application in the IDE by hitting the F9 key.

Specifying Run Parameters

You can specify parameters to be used with the URL used to run the application by using the Parameters option on the Run menu. URL parameters are specified in the following format:

```
?param1=paramvalue1&param2=paramvalue2&param3=paramvalue3
```

You can also specify an anchor to be used with the URL used to run the application:

```
#anchor1
```

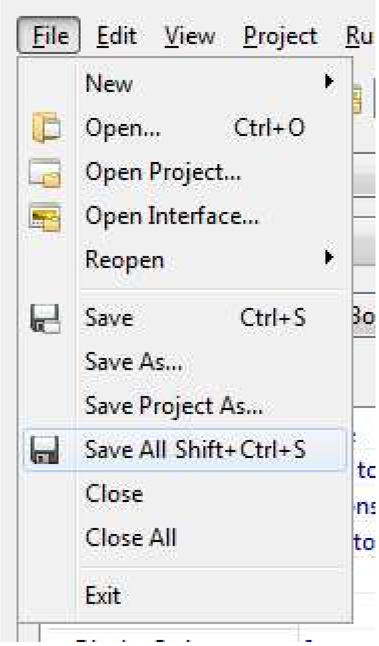
> **Note**
> If specifying both parameters and an anchor, the anchor should be placed after the parameters.

2.8 Saving a Project

Saving Projects

Use the following steps to save a project:

- Click on the **File** option in the main menu. The File menu will open:

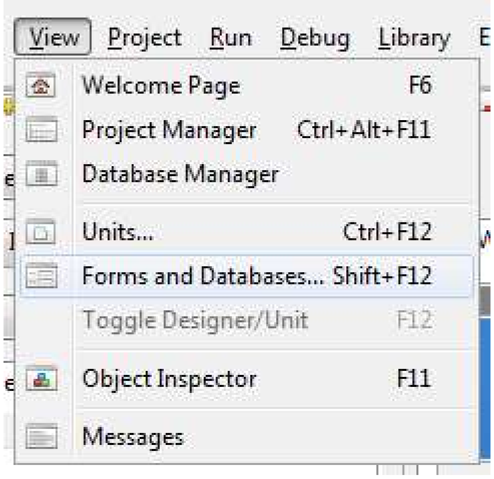


- Click on the **Save All** option in the File menu to save all modified source unit and form files, as well as the project itself.

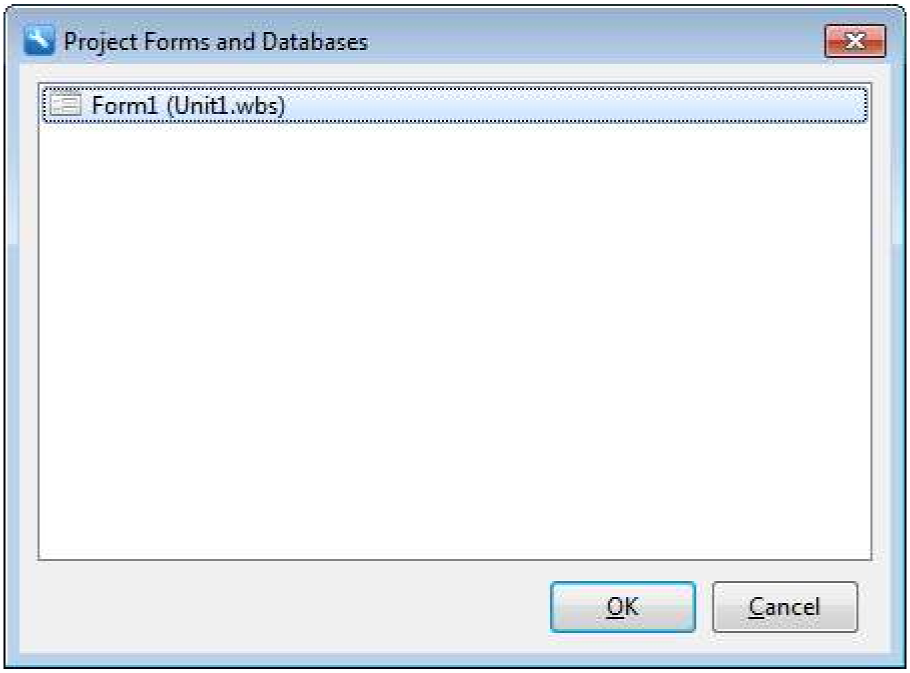
2.9 Viewing Project Forms and Databases

Use the following steps to view a listing of all forms and databases in a project:

- Click on the **View** option in the main menu. The View menu will open:



- Click on the **Forms and Databases** option in the View menu to open the Project Forms and Databases dialog:

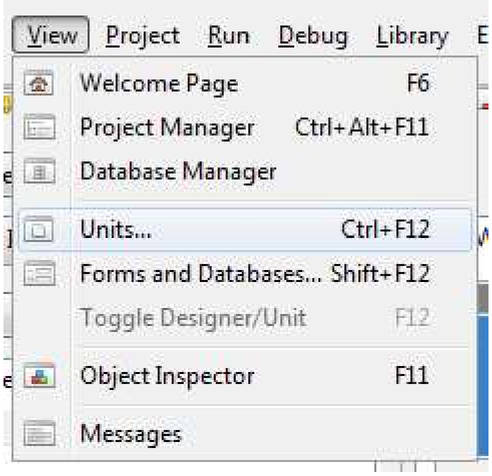


Note
Source units that don't have associated form or database files will not appear in the Project Forms and Databases dialog.

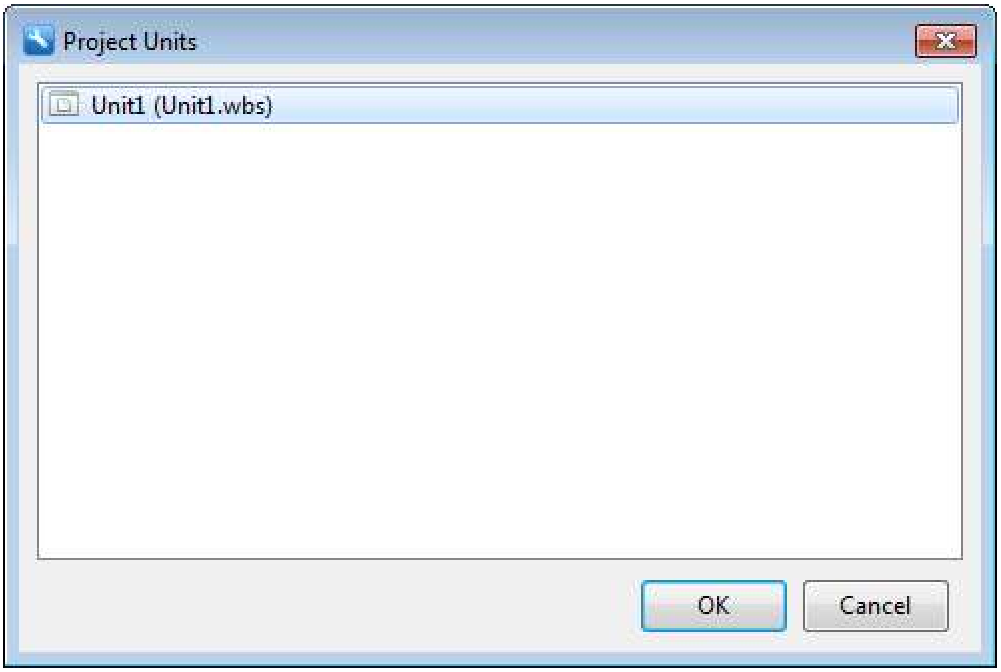
2.10 Viewing Project Units

Use the following steps to view a listing of all source units in a project:

- Click on the **View** option in the main menu. The View menu will open:

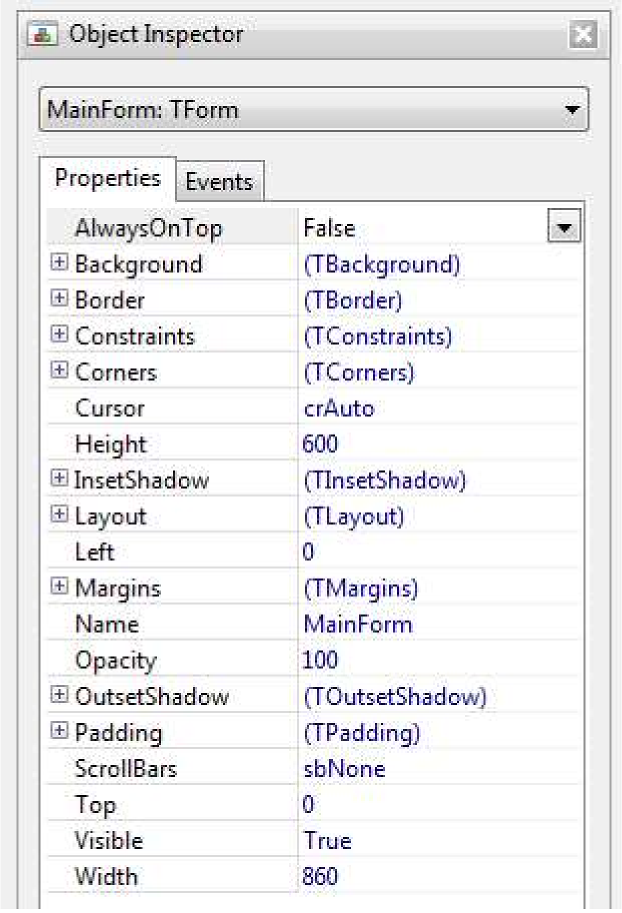


- Click on the **Units** option in the View menu to open the Project Units dialog:



2.11 Using the Object Inspector

The object inspector allows you to modify various properties of each user interface component on a form, giving you complete control over how each component looks and behaves. It is also used to modify the properties of non-visual components that don't actually have any visual presence on a form at run-time but are placed on a form at design-time, such as datasets, timers, and server requests.

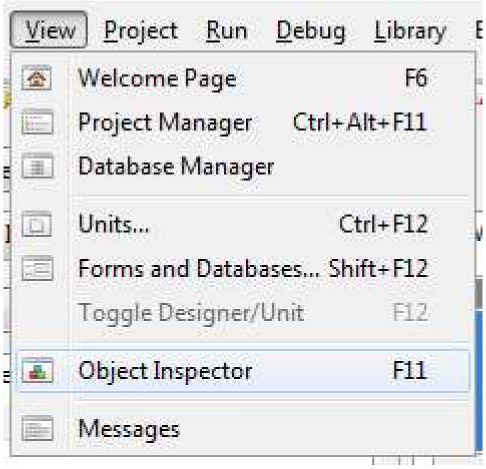


AlwaysOnTop	False
⊞ Background	(TBackground)
⊞ Border	(TBorder)
⊞ Constraints	(TConstraints)
⊞ Corners	(TCorners)
Cursor	crAuto
Height	600
⊞ InsetShadow	(TInsetShadow)
⊞ Layout	(TLayout)
Left	0
⊞ Margins	(TMargins)
Name	MainForm
Opacity	100
⊞ OutsetShadow	(TOutsetShadow)
⊞ Padding	(TPadding)
ScrollBars	sbNone
Top	0
Visible	True
Width	860

Note
The object inspector will only show properties for the currently-selected components on the active form. It will be blank if a form is not active, such as when you are editing a source unit that does not have an associated form, or if there isn't a project open in the IDE. Also, if you have selected more than one component on the active form, the object inspector will only show the properties that are common to all selected components.

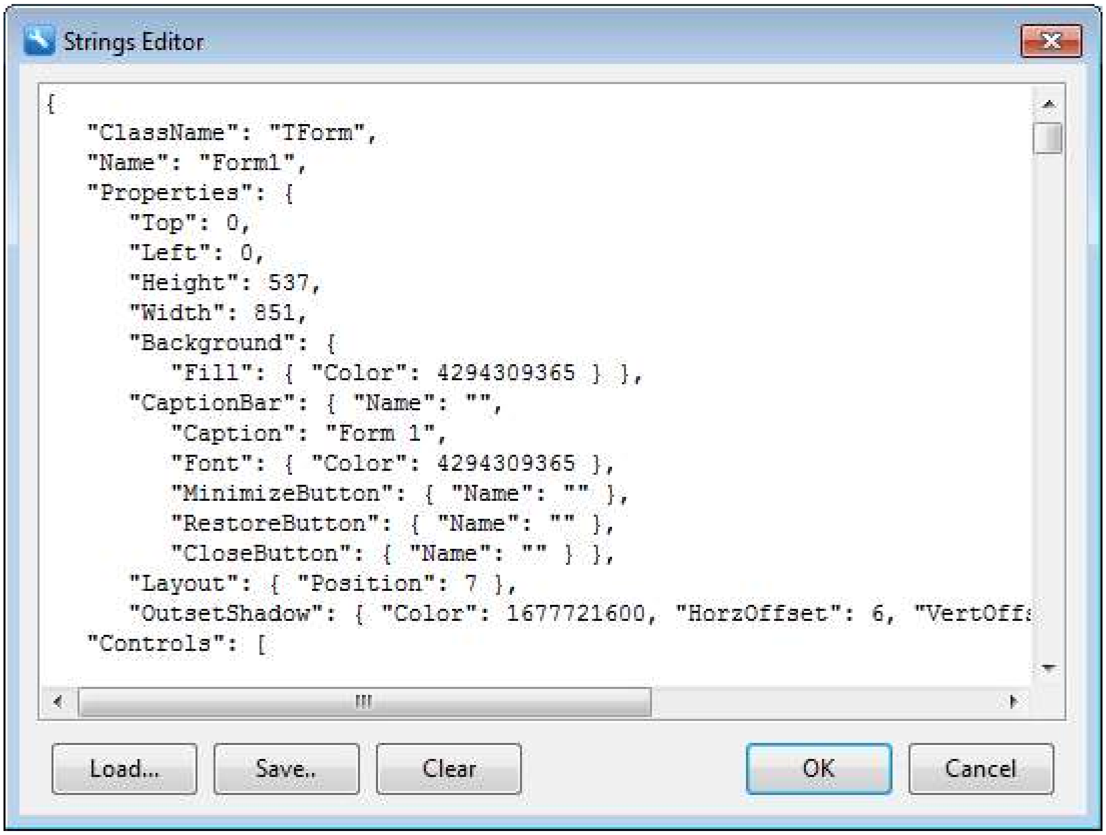
The object inspector consists of a component selection combo box and two pages that represent the properties and events of a component. You can switch between the two by clicking on the appropriate tab at the top of the object inspector.

By default, the object inspector is visible in the IDE. If the object inspector is closed, you can open it by hitting the F11 key, or by using the Object Inspector option on the View menu:



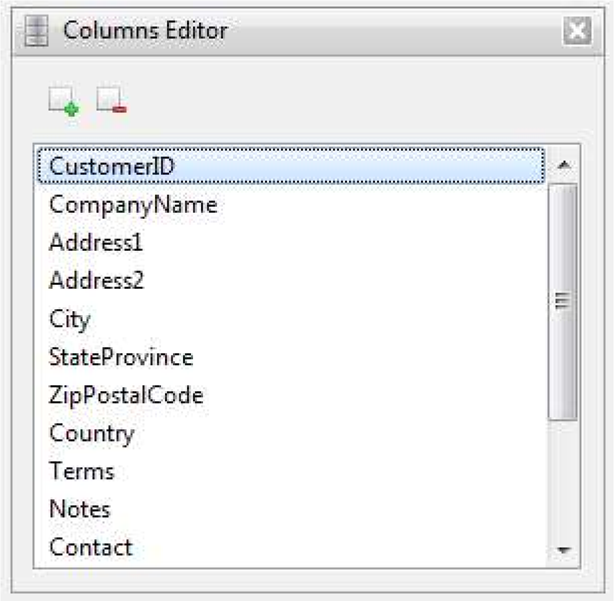
Modifying Properties

To modify any property of a component, make sure that the Properties page is the active page in the object inspector, click on the desired property value, and type in the new value. If applicable, the property may have a special property editor in the form of a drop-down list or dialog that is accessible using a button to the right of the property value. The following is an example of the TStrings property editor for the TMultiLineEdit Lines property:



```
{
   "ClassName": "TForm",
   "Name": "Form1",
   "Properties": {
      "Top": 0,
      "Left": 0,
      "Height": 537,
      "Width": 851,
      "Background": {
         "Fill": { "Color": 4294309365 } },
      "CaptionBar": { "Name": "",
         "Caption": "Form 1",
         "Font": { "Color": 4294309365 },
         "MinimizeButton": { "Name": "" },
         "RestoreButton": { "Name": "" },
         "CloseButton": { "Name": "" } },
      "Layout": { "Position": 7 },
      "OutsetShadow": { "Color": 1677721600, "HorzOffset": 6, "VertOff
   "Controls": [
```

Double-clicking on the property value will also automatically launch the applicable property editor. Properties that represent collections, such as the TDataSet Columns property, will cause an applicable collection editor to be launched below the object inspector:



CustomerID
CompanyName
Address1
Address2
City
StateProvince
ZipPostalCode
Country
Terms
Notes
Contact

Basic information about each property can be found at the bottom of the object inspector.

Modifying Event Handlers

To add, modify, or delete an event handler for a specific component event, make sure that the Events page is the active page in the object inspector, and then click on the desired event. To add a new event handler, or modify an existing event handler, double-click on the event handler name. This will activate the code editor and position you directly on the appropriate event handler code block. If you are adding a new event handler, then the event handler code block will be empty.

Note
If you do not add any code or comments to the new event handler, then it will automatically be removed by the IDE the next time that the source unit and form is saved.

To delete an existing event handler, but keep the event handler code present in the source unit, clear out the event handler name from the event by selecting the entire name and hitting the Delete key. To delete an existing event handler (including the event handler code in the source unit), double-click on the event handler. This will activate the code editor and position you directly on the appropriate event handler code block. Delete all code between the starting begin and end keywords of the event handler code block. The next time the source unit and form is saved, the event handler will automatically be removed by the IDE.

For more information on using the code editor, please see the Code Editor topic.

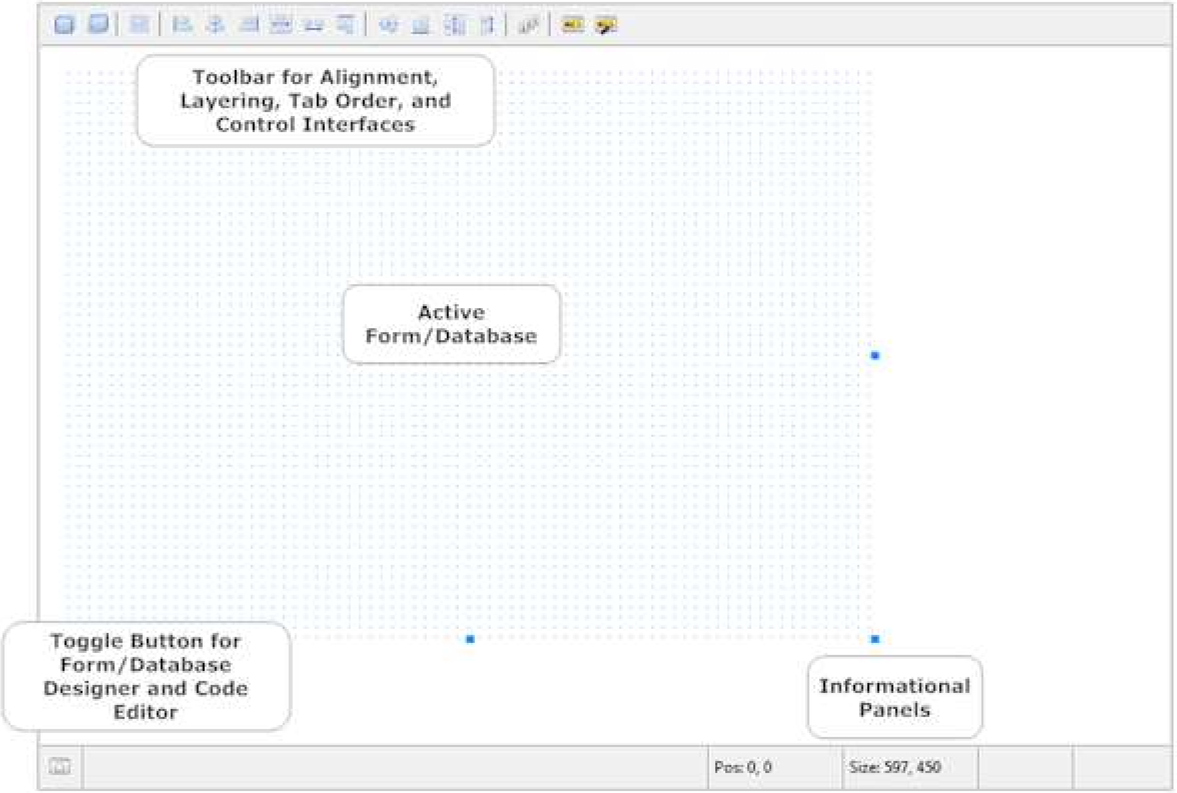
Context-Sensitive Help

You can get context-sensitive help on any property or event in the object inspector by clicking on the desired property or event and hitting the F1 key. For more information on using the help browser, please see the Accessing Help topic.

2.12 Using the Form and Database Designers

One of the first phases of web application development is the design of the user interface and database view(s) for the application. This is accomplished in the IDE by using the WYSIWYG (What You See Is What You Get) form and database designers.

The form and database designers have the following layout:

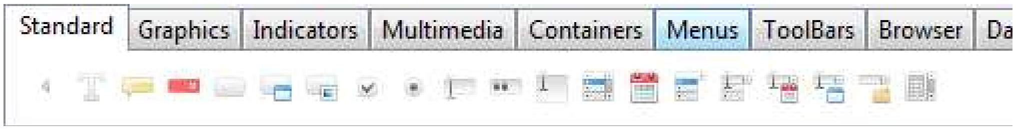

Toolbar for Alignment, Layering, Tab Order, and Control Interfaces
Active Form/Database
Toggle Button for Form/Database Designer and Code Editor
Informational Panels
Pos: 0, 0
Size: 597, 450

The unit of measure used by the designers is the pixel, and the resolution is always assumed to be 96 pixels per inch. All modern web browsers use a virtual resolution of 96 pixels per inch, regardless of the actual resolution on the client machine's display. The web browser automatically handles the translation between the virtual resolution and the display resolution of the client machine.

By default, the designers show a grid to aid with component placement and alignment, and the grid guides (dots) are spaced apart at 8 pixel intervals. Please see the Modifying Environment Options topic for more information on modifying the designer grid properties.

Adding a Component to a Form

The component palette is available at the top of the main IDE window, and reflects all installed components in the component library, organized by their installation category:



The component palette is used to add both non-visual components and visual controls on to the form and database designers for use with your forms and databases. Non-visual components are represented visually at design-time, but are actually non-visual components at runtime.

> **Note**
> The database designer only allows for non-visual components to be placed on the designer surface, and the visual size of the database instance in the designer is exclusively a design-time property.

To see more information about a particular component, hover the mouse over the component icon. The IDE will display the name of the component and the unit in which it resides in a tooltip window.

To add a non-visual component or visual control to the active form or database in the designer, click on the desired component/control on the component palette, and then click on the active form or database's client area. A form's client area is the area inside of the borders and caption bar (if present), whereas the database's client area is the database's entire designer space.

Selecting a Component

To select a single component in the form and database designers, click on the desired component with the left mouse button. To select more than one component, hold down the Shift key while clicking on the desired components with the left mouse button. Selecting multiple components is desirable when one wants to resize or align multiple components at the same time to ensure that their placement or size is uniform, or when one wants to copy and paste a group of controls or components.

> **Note**
> Any time you hover the mouse over any component on the active form or database, tooltip information will be displayed about the component, including the name and position/size.

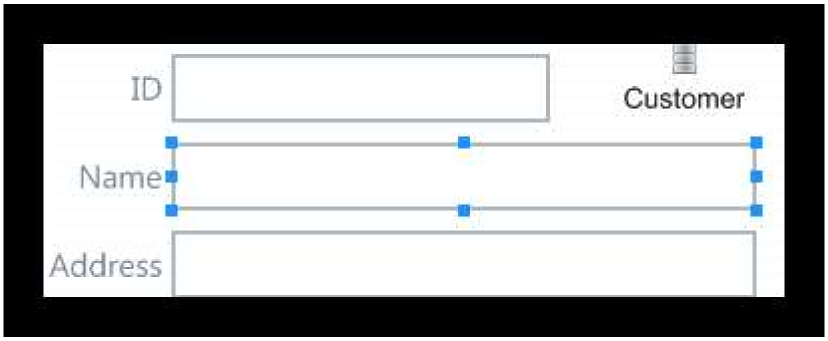
You can also use the mouse to directly select a group of components using a lasso:

- If the group of components are placed on the form or database itself, you can click and hold down the left mouse button to begin the selection. Then, while keeping the left mouse button down, move the mouse to lasso the desired component(s).

- If the group of components are placed on a sub-container (such as a panel), you can click and hold down the left mouse button, while also pressing the Ctrl key, to begin the selection. Then, while keeping the left mouse button and Ctrl key down, move the mouse to lasso the desired component(s).

Resizing a Component

Once a component has been placed on the active form or database's client area, you will see that the component will have designer handles on all four sides and corners of the component:



These designer handles can be used to change the origin and size of a component on the form or database. To accomplish this, click on a designer handle with the left mouse button, hold the left mouse button down, and drag the designer handle in the desired direction. You can also use the keyboard to resize a component by holding down the Shift key while using the up, down, right, and left arrow keys to resize the component on a pixel-by-pixel basis.

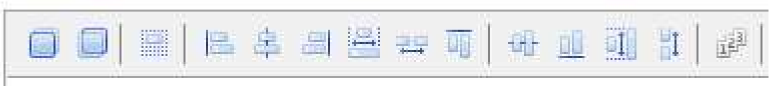
> **Note**
> Certain components may have constraints on how tall/wide they can be, and non-visual components cannot be resized at all. In such cases, attempts to resize the component will result in the component size not exceeding the constraints imposed by the type of component. Also, you cannot use the left mouse button to resize components when multiple components are selected. In such cases, you can only use the keyboard to do so (Shift+Arrow Keys).

Moving a Component

To move a component, click on the component with the left mouse button, hold the mouse button down, and drag the component to the desired location on the form or database. You can also use the keyboard to move a component by holding down the Ctrl key while using the up, down, right, and left arrow keys to move the component on a pixel-by-pixel basis. Both of these techniques also work when multiple components are selected.

Component Layout and Alignment

The layout toolbar on the form and database designers can be used to adjust the alignment, layering (send to back/bring to front), and tab ordering of components on the active form or database:



Each layout toolbar button has tooltip help that explains the purpose of the button.

Deleting a Component

To delete a component, select the desired component in the form or database designer and hit the Delete key. This will also work when multiple components are selected.

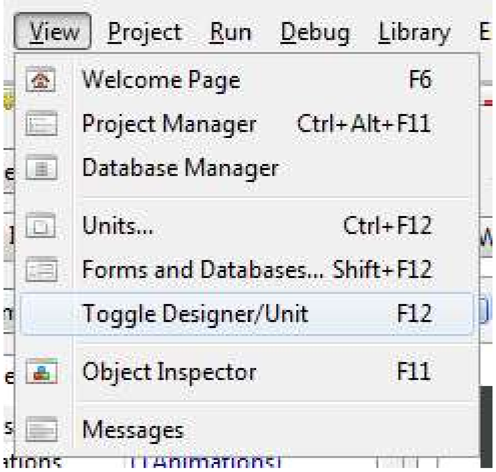
Warning
Undo functionality is currently not available for the form and database designers, so any modifications or deletions of components cannot be undone. Please be careful when deleting components to ensure that one does not lose a lot of hard work. If you do accidentally delete a component from a form or database, you can fix the issue by simply closing the form or database without saving the modifications, and then re-opening the form or database. However, this depends upon how much other work has been done to the form or database since the last save point, so it is wise to save your modifications on a regular basis.

Default Event Handlers

If you double-click on a component in the form and database designers, a new event handler will be created for the default event property for the component. For most visual or bindable controls, the default event property is the OnClick or OnChange event. Please see the Events topic in the Language Reference for more information on default events.

Toggling Between the Code Editor and Designer

In order to toggle between the code editor and the designer, hit the F12 key, click on the toggle button at the bottom left-hand corner of the code editor and designer, or use the Toggle Designer/Unit option on the View menu:

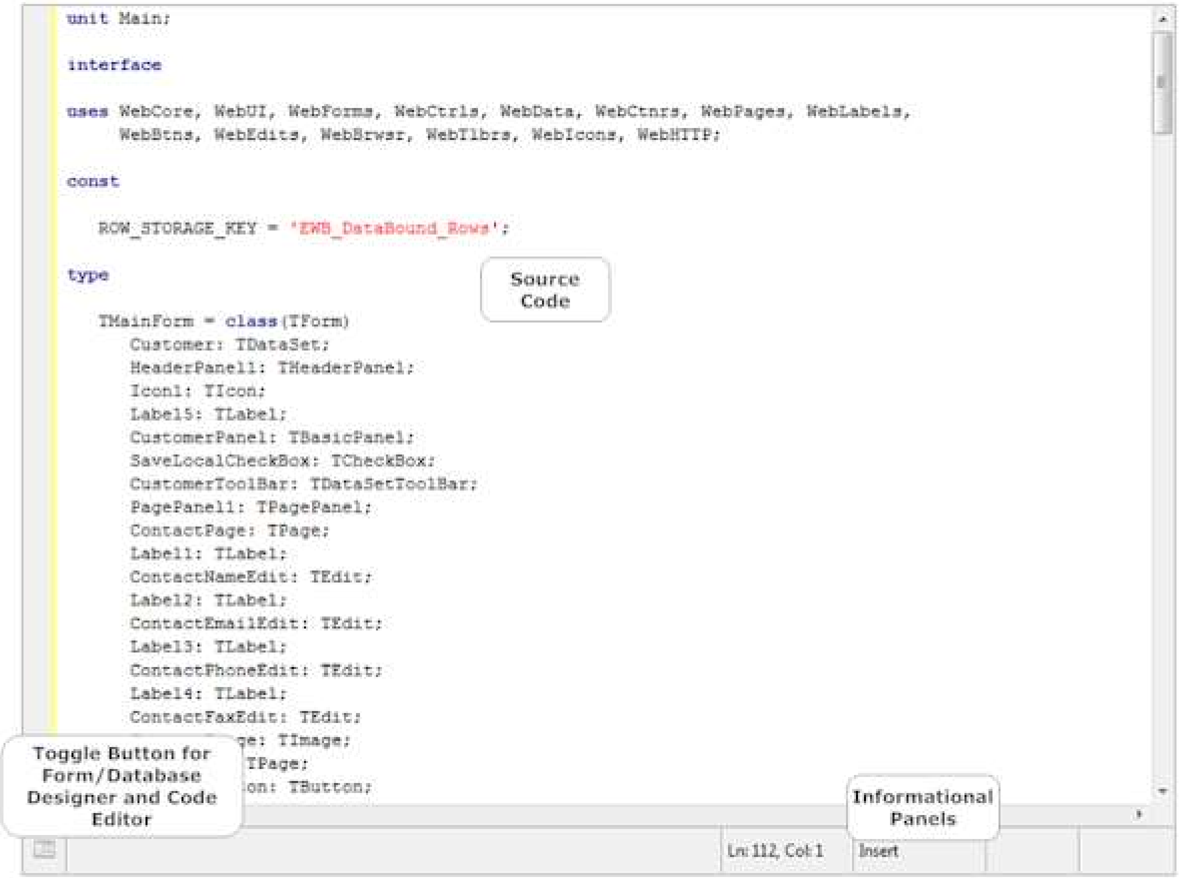


Context-Sensitive Help

You can get context-sensitive help on any component in the form and database designers by selecting the desired component and hitting the F1 key. For more information on using the help browser, please see the Accessing Help topic.

2.13 Using the Code Editor

While the form and database designers handle the user interface design of the application, the code editor is where the actual functionality behind a form or database is implemented. The code editor has the following layout:



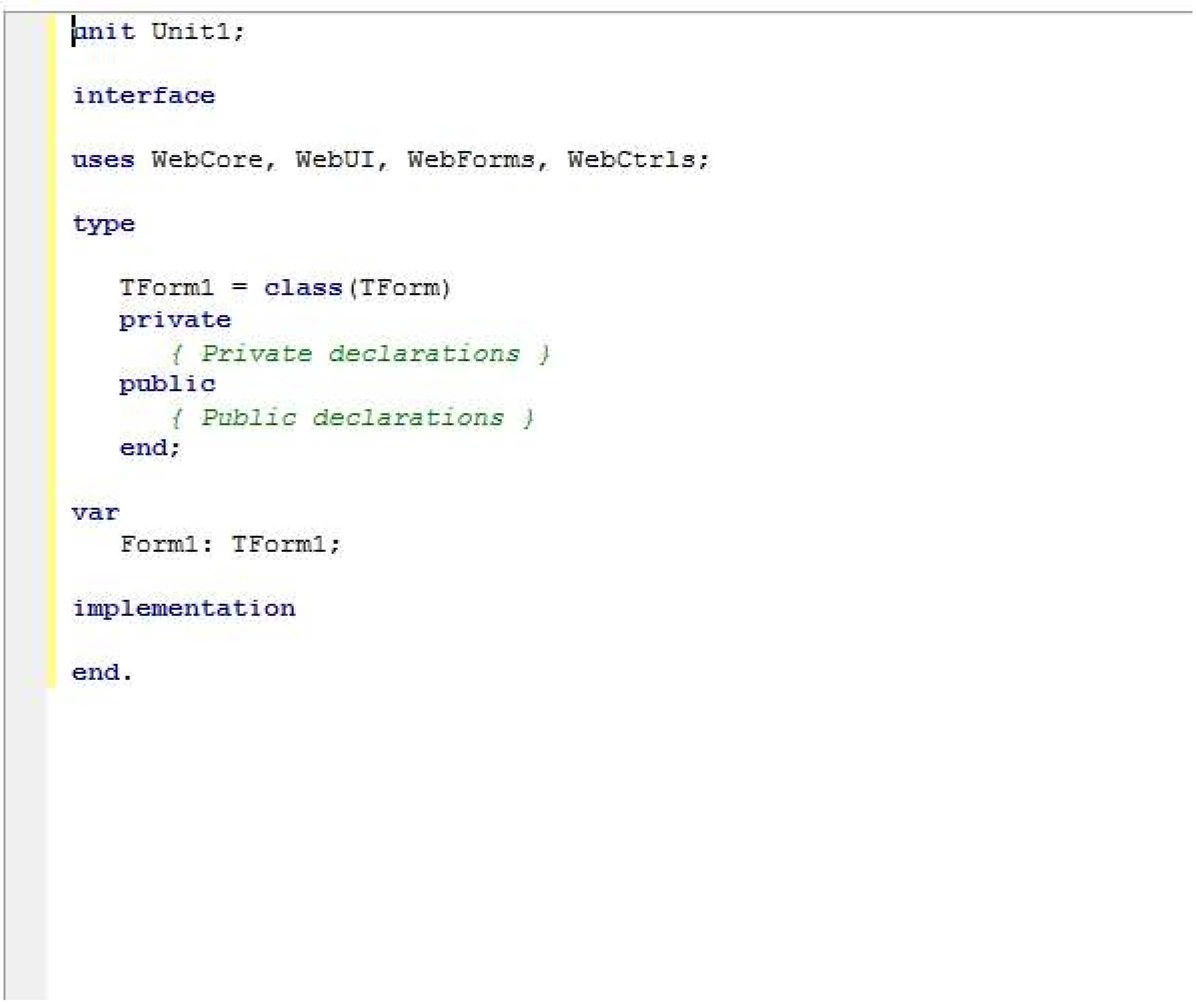
```
unit Main;

interface

uses WebCore, WebUI, WebForms, WebCtrls, WebData, WebCtnrs, WebPages, WebLabels,
     WebBtns, WebEdits, WebBrwsr, WebTlbrs, WebIcons, WebHTTP;

const

   ROW_STORAGE_KEY = 'EWB_DataBound_Rows';

type

   TMainForm = class(TForm)
      Customer: TDataSet;
      HeaderPanel1: THeaderPanel;
      Icon1: TIcon;
      Label5: TLabel;
      CustomerPanel: TBasicPanel;
      SaveLocalCheckBox: TCheckBox;
      CustomerToolBar: TDataSetToolBar;
      PagePanel1: TPagePanel;
      ContactPage: TPage;
      Label1: TLabel;
      ContactNameEdit: TEdit;
      Label2: TLabel;
      ContactEmailEdit: TEdit;
      Label3: TLabel;
      ContactPhoneEdit: TEdit;
      Label4: TLabel;
      ContactFaxEdit: TEdit;
                ge: TImage;
                 TPage;
                on: TButton;
```

Ln: 112, Col: 1 | Insert

Warning
Although all Unicode characters are supported in the code editor, certain double-wide characters in languages such as Chinese and Japanese cannot be displayed/edited properly at this time.

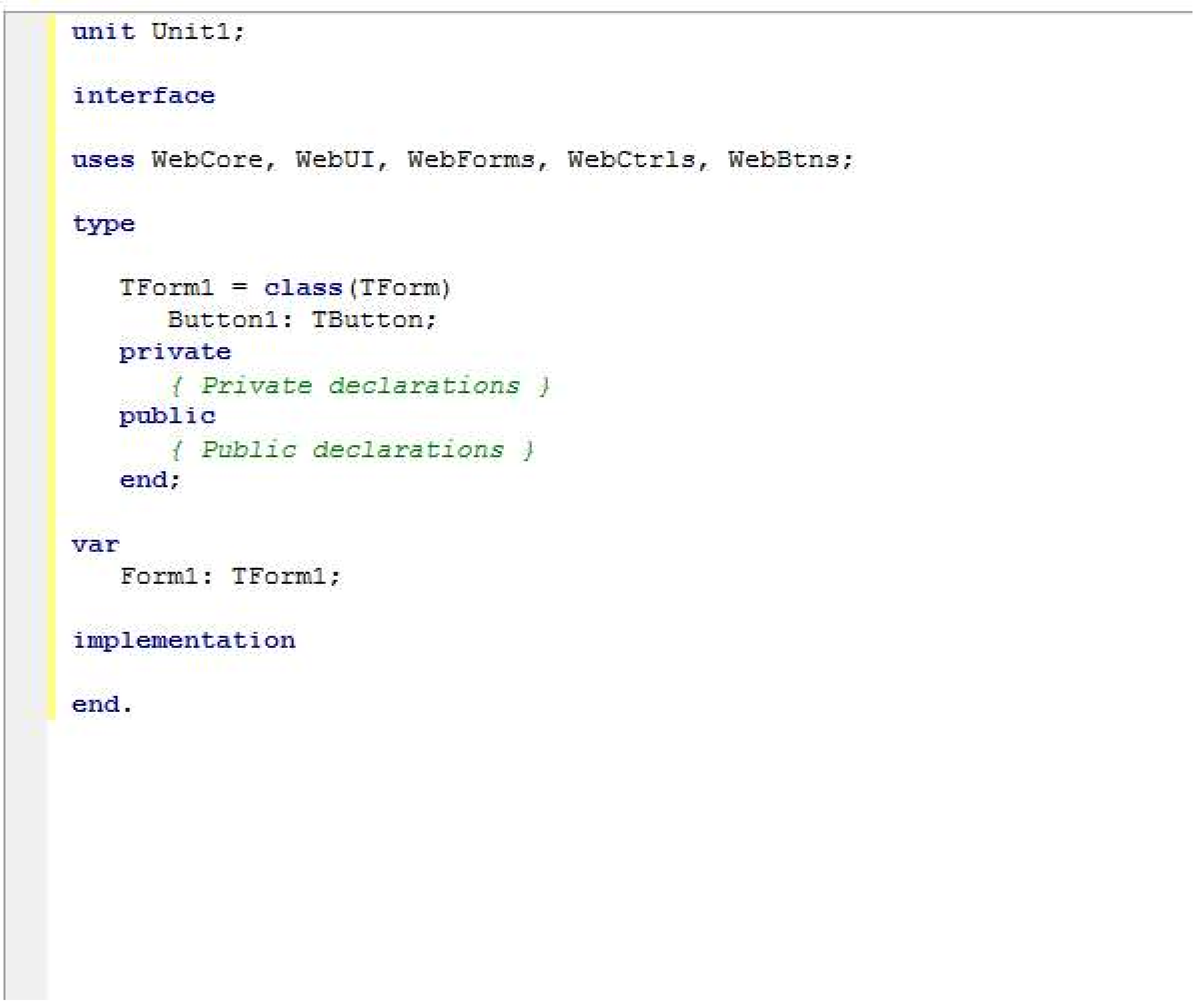
Automatic Code Updates

All component additions, modifications, and deletions are automatically reflected in the code editor by the IDE. For example, the following is the code editor showing the source unit of a new form:



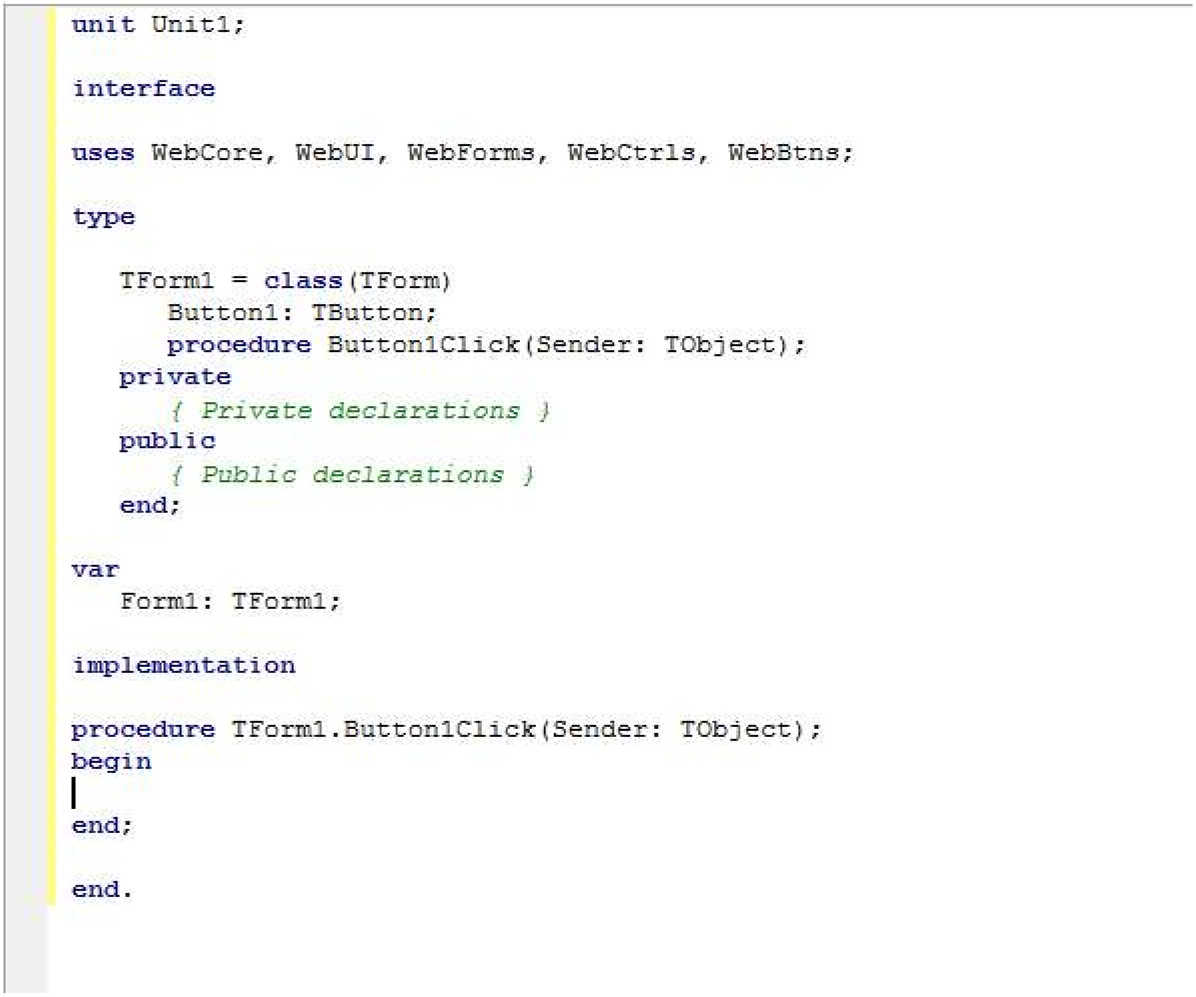
```
unit Unit1;

interface

uses WebCore, WebUI, WebForms, WebCtrls;

type

   TForm1 = class(TForm)
   private
      { Private declarations }
   public
      { Public declarations }
   end;

var
   Form1: TForm1;

implementation

end.
```

The following is the same source unit in the code editor after adding a TButton component.



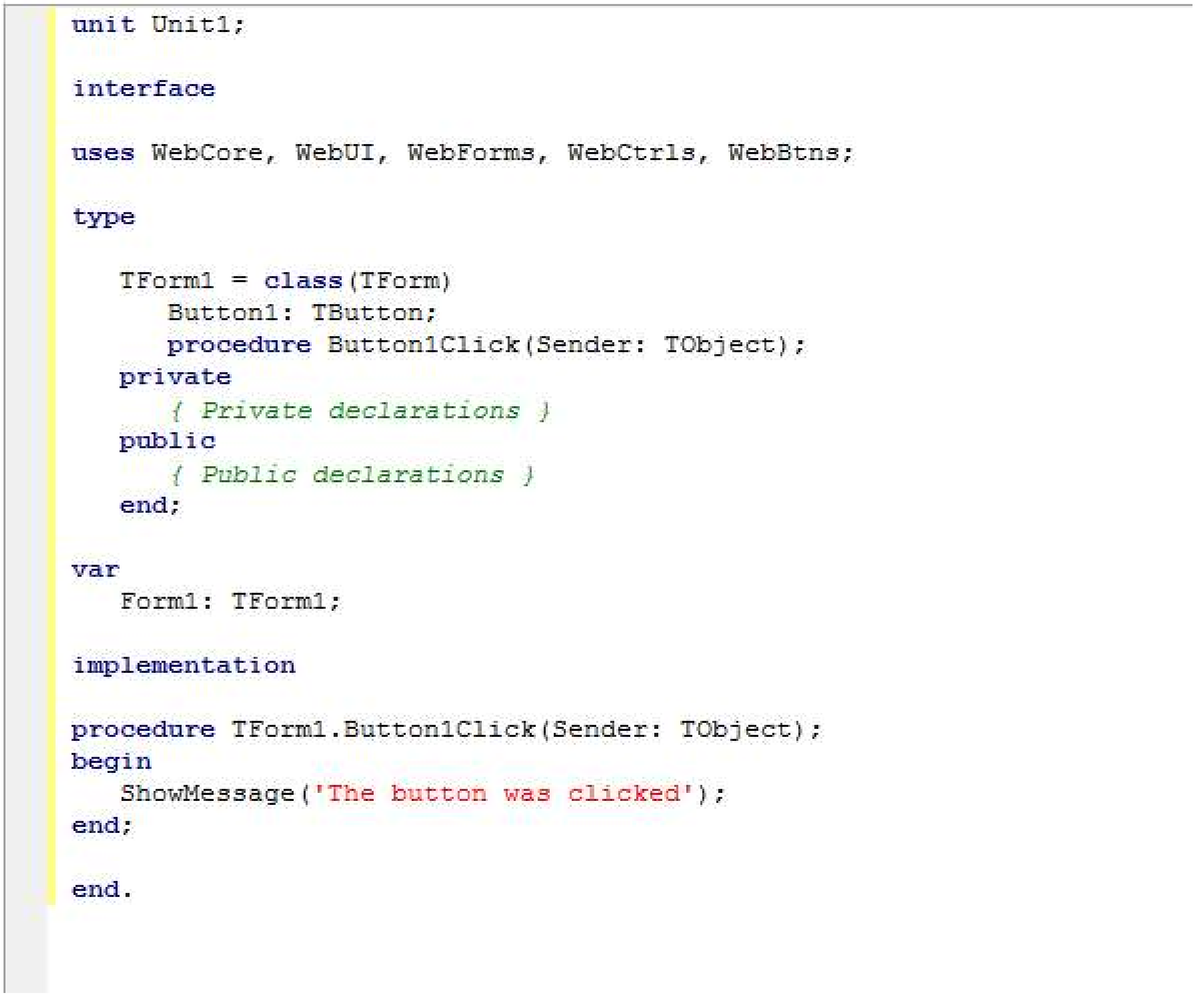
```
unit Unit1;

interface

uses WebCore, WebUI, WebForms, WebCtrls, WebBtns;

type

   TForm1 = class(TForm)
      Button1: TButton;
   private
      { Private declarations }
   public
      { Public declarations }
   end;

var
   Form1: TForm1;

implementation

end.
```

As you can see, the IDE automatically updated the source unit to include the proper declaration for the newly-added TButton component called Button1. If you then double-click on the Button1 component in the form designer, the source unit will look like the following:



```
unit Unit1;

interface

uses WebCore, WebUI, WebForms, WebCtrls, WebBtns;

type

  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

procedure TForm1.Button1Click(Sender: TObject);
begin

end;

end.
```

Again, the IDE has automatically updated the source unit to include an empty event handler for the TButton OnClick event. If you add code to the empty event handler, this code will then be executed when the button is clicked. For example, let's add a call to the ShowMessage procedure to display a message to the user:



```
unit Unit1;

interface

uses WebCore, WebUI, WebForms, WebCtrls, WebBtns;

type

   TForm1 = class(TForm)
      Button1: TButton;
      procedure Button1Click(Sender: TObject);
   private
      { Private declarations }
   public
      { Public declarations }
   end;

var
   Form1: TForm1;

implementation

procedure TForm1.Button1Click(Sender: TObject);
begin
   ShowMessage('The button was clicked');
end;

end.
```

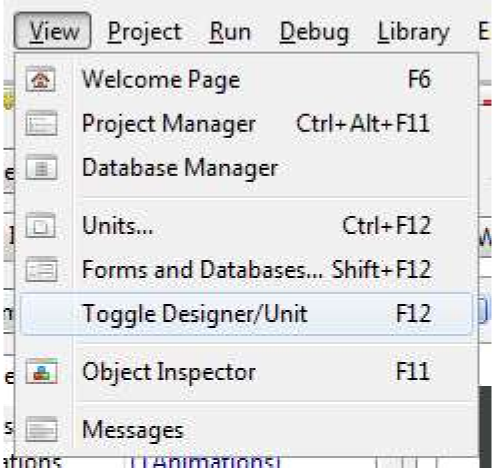
If you do not define any code or comments between the begin and end keywords that define the event handler code block, the IDE will automatically remove the event handler completely from the source unit the next time the source unit and form is saved.

Toggling Between the Code Editor and Designer

There are three ways to toggle between the code editor and the designer:

- Hitting the F12 key
- Clicking on the toggle button at the bottom left-hand corner of the code editor and designer

- Using the Toggle Form/Unit option on the View menu:



Key Mappings

The following key mappings are active in the code editor. Unless indicated otherwise, holding down the Shift key while pressing any of the keys that move the cursor position will cause any source code between the original and the final cursor position to be selected.

Keys	Action
Up Arrow	Moves the cursor to the previous line in the source code.
Down Arrow	Moves the cursor to the next line in the source code.
Page Up	Moves the cursor to the previous page in the source code.
Page Down	Moves the cursor to the next page in the source code.
Home	Moves the cursor to the start of the current source code line.
End	Moves the cursor to the end of the current source code line.
Left Arrow	Moves the cursor to the previous character on the current source code line.
Right Arrow	Moves the cursor to the next character on the current source code line, or to the next line if at the end of the current source code line.
Enter	Inserts a new line at the current cursor position.
Insert	Toggles the insert/overwrite mode for the keyboard.
Shift-Insert	Pastes the source code contents of the clipboard, if any, into the current cursor position.
Delete	Deletes the character at the cursor position.
Shift-Delete	Copies the currently-selected source code to the clipboard and deletes the source code from the source code ("cut" operation).

Backspace	Deletes the character right before the cursor position. If the cursor is at the start of a source code line, then the current source code line is moved to the end of the previous source code line (if present).
Tab	Inserts <tab size> spaces at the current cursor position, if the keyboard is in insert mode, or moves the current cursor position by <tab size> spaces if the keyboard is in overwrite mode. Please see the Modifying Environment Options topic for more information on modifying the tab size used by the code editor.
Shift-Tab	Removes <tab size> spaces working back from the current cursor position, if the keyboard is in insert mode, or moves the current cursor position to the left by <tab size> spaces if the keyboard is in overwrite mode. Please see the Modifying Environment Options topic for more information on modifying the tab size used by the code editor.
Ctrl-Home	Moves the cursor to the first source code line.
Ctrl-End	Moves the cursor to the last source code line.
Ctrl-Page Up	Moves the cursor to the first visible line on the current page.
Ctrl-Page Down	Moves the cursor to the last visible line on the current page.
Ctrl-Left Arrow	Moves the cursor to the start of the previous token in the source code.
Ctrl-Right Arrow	Moves the cursor to the start of the next token in the source code.
Ctrl-Up Arrow	Scrolls the code editor window up by one line.
Ctrl-Right Arrow	Scrolls the code editor window down by one line.
Ctrl-Enter	Opens the unit name or control interface name at the cursor position. If text is selected, then the selected text will be used first for searching for a valid unit name or control interface. If no text is selected, or the selected text does not represent a valid unit or control interface name, then the editor will parse the current token and use it instead.
Ctrl-/	Comments and un-comments (toggle) the current source code line.
Ctrl-Insert	Copies the currently-selected source code to the clipboard.
Ctrl-Backspace	Deletes the token right on, or right before, the current cursor position. If the cursor is at the start of a source code line, then the current source code line is moved to the end of the previous source code line (if present).
Ctrl-A	Selects all source code in the code editor.
Ctrl-C	Copies the currently-selected source code to the clipboard.

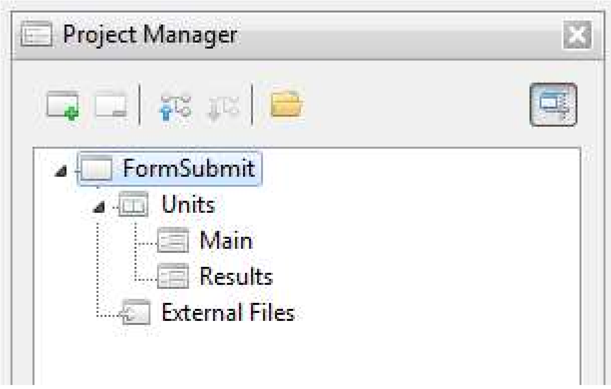
Ctrl-I	Indents the current source code line by <tab size> spaces. Please see the Modifying Environment Options topic for more information on modifying the tab size used by the code editor.
Ctrl-N	Inserts a new line at the current cursor position.
Ctrl-T	Deletes the token at the current cursor position.
Ctrl-U	Un-indents the current source code line by <tab size> spaces. Please see the Modifying Environment Options topic for more information on modifying the tab size used by the code editor.
Ctrl-V	Pastes the source code contents of the clipboard, if any, into the current cursor position.
Ctrl-X	Copies the currently-selected source code to the clipboard and deletes the source code from the source code ("cut" operation).
Ctrl-Y	Deletes the current source code line.
Shift-Ctrl-Y	Deletes the source code from the current cursor position to the end of the current source code line.
Ctrl-Z	Reverses the last edit or find operation performed on the source code ("undo" operation).
Shift-Ctrl-Z	Replays the last edit or find operation on the source code that was reversed ("redo" operation).
Shift-Ctrl-Down Arrow	Moves the cursor from the class definition of a method to the implementation of the method.
Shift-Ctrl-Up Arrow	Moves the cursor from the implementation of a method to its class definition.

Context-Sensitive Help

You can get context-sensitive help on any keyword or identifier in the code editor by positioning the cursor over the desired keyword or identifier and hitting the F1 key. For more information on using the help browser, please see the Accessing Help topic.

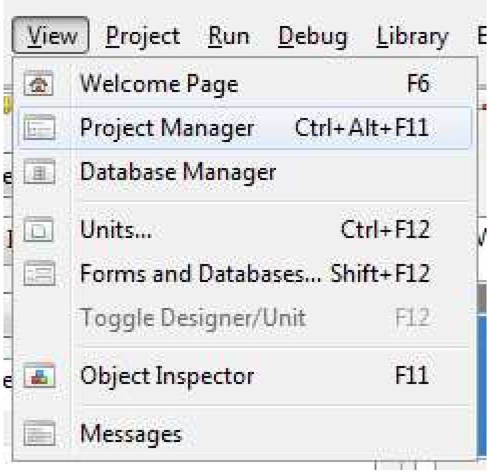
2.14 Using the Project Manager

The project manager provides a quick and easy-to-use interface to the contents of a project, including all source units (code-behind units for forms and databases, and code-only units) and external files like images or Javascript source files.



Note
 Simply adding an external JavaScript source file to a project is insufficient for actually referencing such external code from within an Elevate Web Builder application. You must also define an external interface to the classes, functions, procedures, and variables that you wish to reference in your application code. For more information on defining external interfaces, please see the External Interfaces topic.

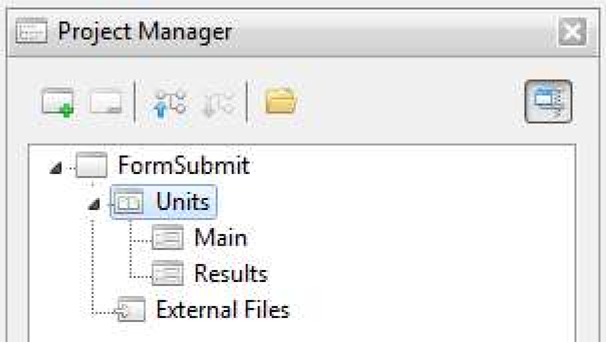
By default, the project manager is visible in the IDE. If the project manager is closed, you can open it by holding down the Ctrl and Alt keys and hitting the F11 key, or by using the Project Manager option on the View menu:



Adding an Existing Source Unit to a Project

Use the following steps to add an existing source unit to a project using the project manager:

- Click on the **Units** node of the project contents tree.



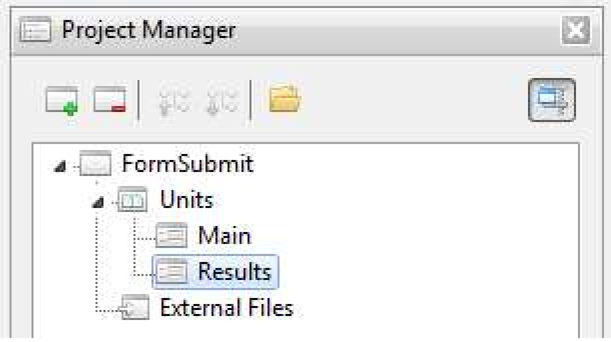
- Click on the Add button in the project manager toolbar:

- A Windows file open dialog will appear. Navigate to, and select, the existing source unit that you wish to add to the project. Click on the Open button in the Windows file open dialog to complete adding the source unit to the project.

Removing a Source Unit from a Project

Use the following steps to remove a source unit from a project using the project manager:

- Click on the name of the source unit that you wish to remove:



- Click on the Remove button in the project manager toolbar:

- A confirmation dialog will be displayed, asking you to confirm the removal of the source unit from the project. Click on the Yes button to continue, or the No button to cancel the removal.

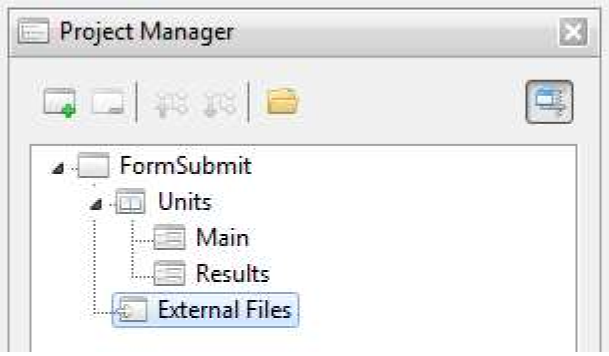
Note

Removing a source unit from a project does **not** delete the actual source unit file on disk. It only removes the reference to the source unit from the project source file (.wbp) so that it will not be considered part of the project anymore.

Adding an Existing External File to a Project

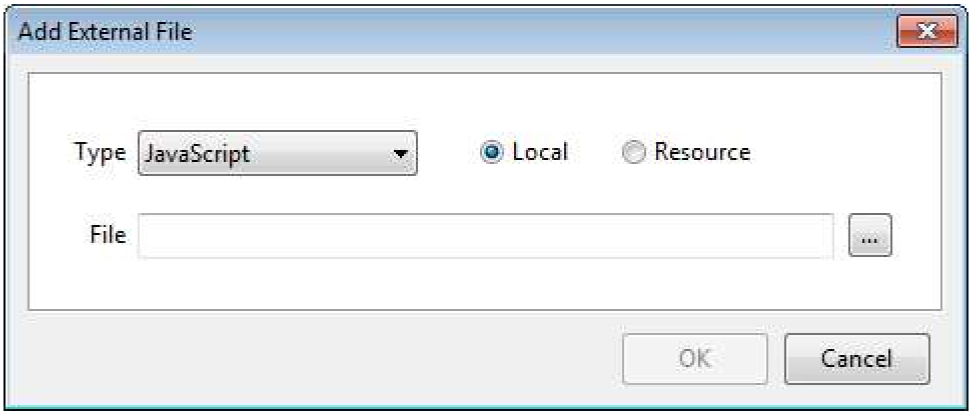
Use the following steps to add an existing external JavaScript file to a project using the project manager:

- Click on the **External Files** node of the project contents tree.



- Click on the Add button in the project manager toolbar:

- The Add External File dialog will appear:



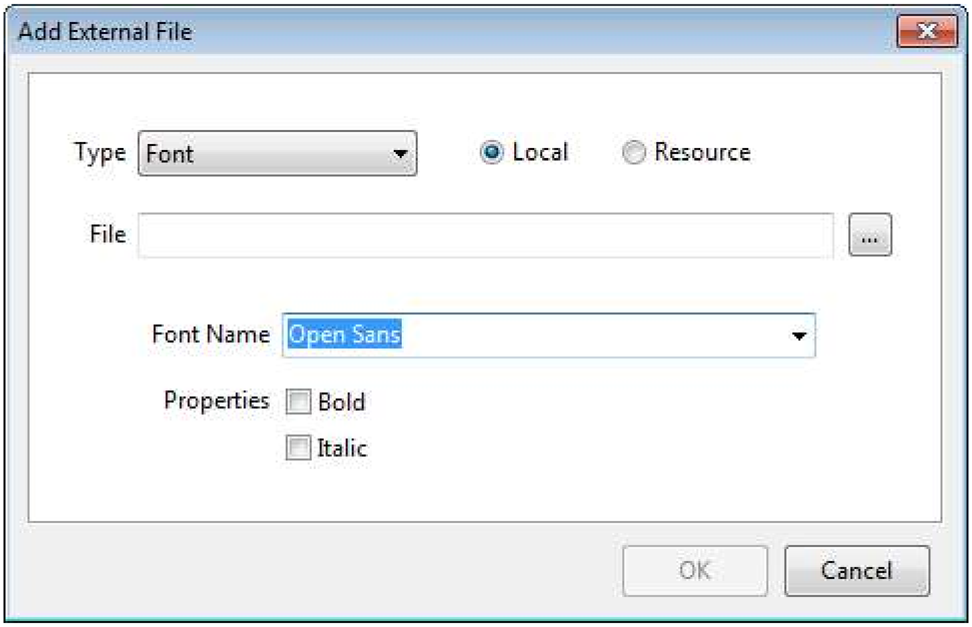
Type JavaScript ● Local ○ Resource

File

Select the type of external file to add using the Type combo box. If the file is a local file (the default), then leave the Local radio button selected and specify the local file name using the File edit control and/or the file selection button to the right of the edit control. If the file is an http resource, then select the Resource radio button and specify the URL for the resource using the File edit control. Click the OK button when you are done specifying the external file to add.

> **Note**
> When adding an external local file, Elevate Web Builder will automatically convert any absolute path specified for the external file to a path that is relative to the current project directory.

If you select Font as the type of external file to add, the Add External File dialog will change to look like the following:

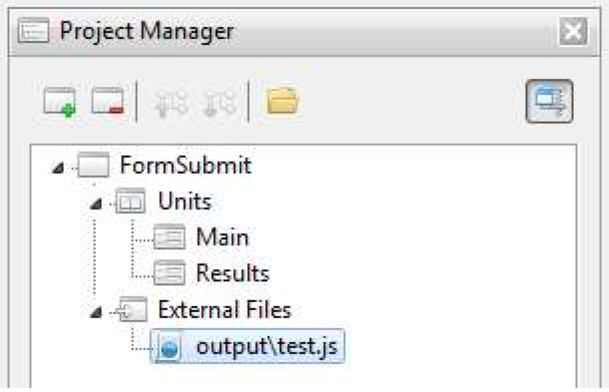


This additional information is necessary to ensure that the proper font linking information is added to the project's HTML loader file during compilation, and to ensure that the proper font is selected at runtime. Use the Font Name combo box to select or enter the name of the font that should be used at runtime, and the Bold and Italic check boxes to specify if the font is a bold or italic version of the font.

Removing an External File from a Project

Use the following steps to remove an external file from a project using the project manager:

- Click on the name of the external file that you wish to remove:



- Click on the Remove button in the project manager toolbar:

- A confirmation dialog will be displayed, asking you to confirm the removal of the external file from the project. Click on the Yes button to continue, or the No button to cancel the removal.

Note
Removing an external file from a project does **not** delete the actual external file on disk. It only removes the reference to the external file from the project configuration file (.wbc) so that it will no longer be considered part of the project.

Opening the Project Folder

You can quickly open the project folder for browsing in the operating system by using the project folder toolbar button in the project manager.

To open the project folder:

- Click on the Project Folder button in the project manager toolbar:

Quick Compiler Settings for a Project

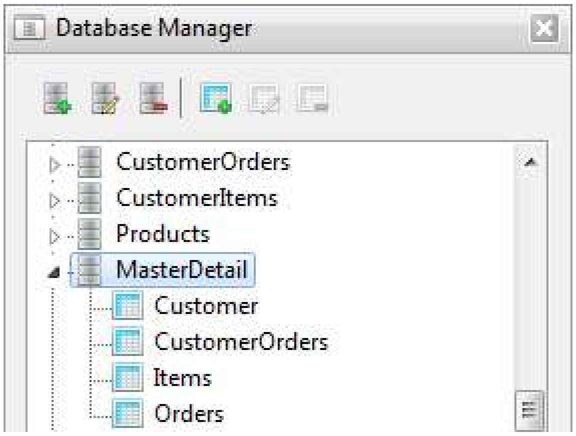
You can toggle certain compilation settings quickly by using the project compiler options toolbar in the project manager.

To toggle the compressed output compilation setting:

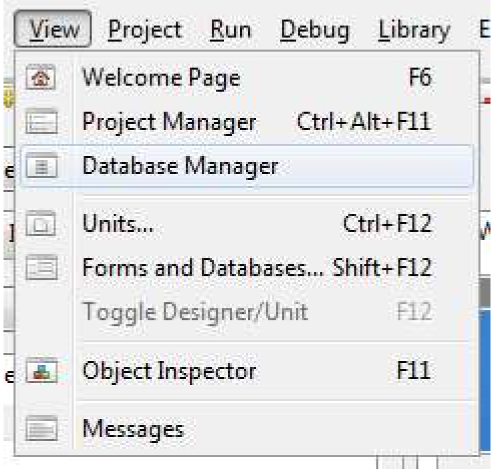
- Click on the Compressed Output button in the project manager toolbar:

2.15 Using the Database Manager

The database manager provides a quick and easy-to-use interface to the databases and datasets defined in the IDE. The databases and datasets that are defined in the database manager are only available to the internal web server embedded in the IDE, and are a way of automating the usage of databases and datasets across multiple projects. The outer nodes in the database manager represent the defined databases, with all datasets within a given database defined as child nodes of each database node.



By default, the database manager is visible in the IDE. If the database manager is closed, you can open it by using the Database Manager option on the View menu:



Using Databases and DataSets in Projects

A database defined in the database manager can be used to create a database in a project by dragging the database from the database manager and dropping it into the project manager for the currently-opened project. When the database is dropped on or within the **Units** node of the project manager, a new TDatabase (or descendant) instance will be created for the project, along with an associated unit, and all of the defined datasets for the database will automatically be created as TDataSet instances in the new database instance.

A dataset defined in the database manager can be used to create a new dataset in the project by dragging the dataset from the database manager and dropping it on an open form or database designer in the currently-opened project. When the dataset is dropped on the form or database designer, a TDataSet instance will be created as a component of the form or database and all of the columns in the dataset will automatically be created in the new TDataSet instance.

Adding a New Database

Use the following steps to add a new database using the database manager:

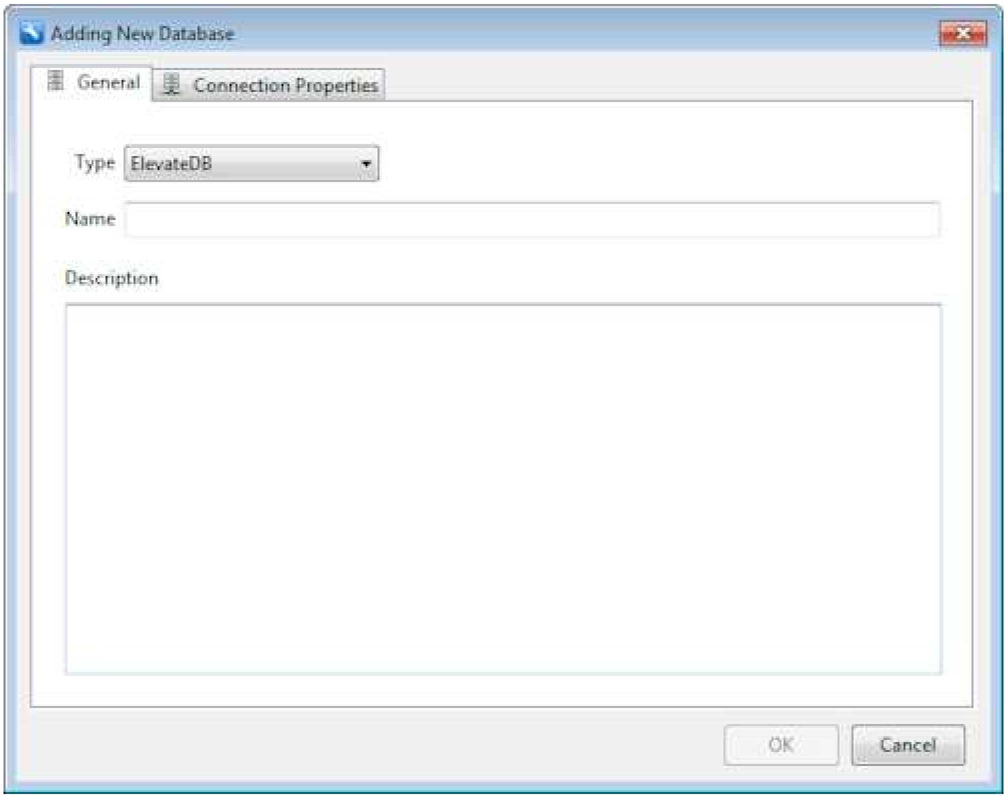
- Click on the Add Database button in the database manager toolbar:

- The database editor dialog will appear. Please refer to the next section for information on defining the database.

Defining a Database

The database editor dialog consists of 2 pages:

- **General** - the database engine/server type, the name of the database, and the description.



Adding New Database

Type ElevateDB

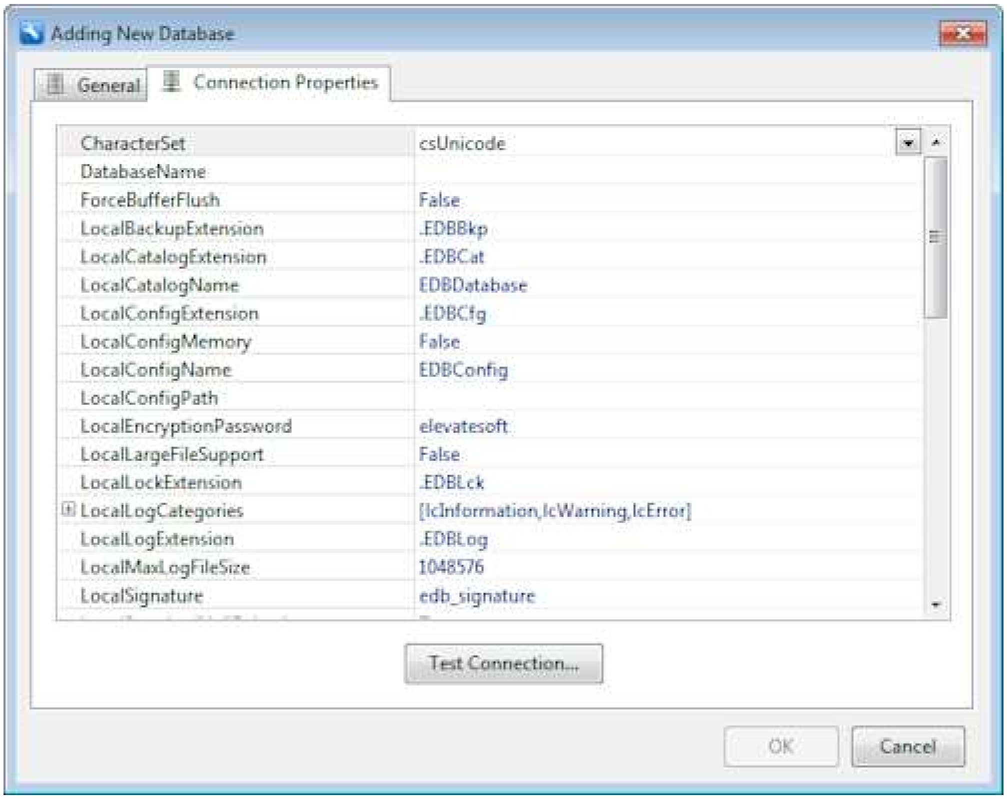
Name

Description

Currently, the following database engines are supported:

ElevateDB
DBISAM
ADO (includes OLEDB/ODBC)

- **Connection Properties** - the name/location of the database and other configuration properties essential to establishing a proper connection to the desired database. The options on this page are specific to the database engine selected on the first page.



CharacterSet	csUnicode
DatabaseName	
ForceBufferFlush	False
LocalBackupExtension	.EDBBkp
LocalCatalogExtension	.EDBCat
LocalCatalogName	EDBDatabase
LocalConfigExtension	.EDBCfg
LocalConfigMemory	False
LocalConfigName	EDBConfig
LocalConfigPath	
LocalEncryptionPassword	elevatesoft
LocalLargeFileSupport	False
LocalLockExtension	.EDBLck
⊞ LocalLogCategories	[lcInformation,lcWarning,lcError]
LocalLogExtension	.EDBLog
LocalMaxLogFileSize	1048576
LocalSignature	edb_signature

Once the connection properties are set, you can use the Test Connection button to verify that everything is set properly. Please see your database engine manual/documentation for more information on the proper value for each property setting.

- Once you have properly set the connection properties and successfully tested the connection to the database, click on the OK button to close the database dialog and save the database.

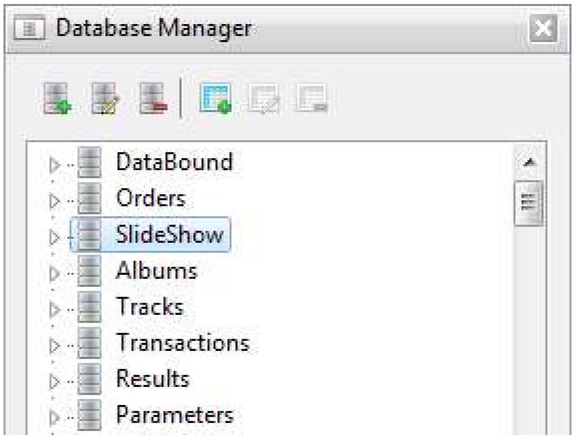
Editing an Existing Database

To edit an existing database using the database manager, simply double-click on the desired database in the list of databases in the database manager. The database editor dialog will then appear, and you can use it to modify the database accordingly.

Removing a Database

Use the following steps to remove a database using the database manager:

- Click on the name of the database that you wish to remove:



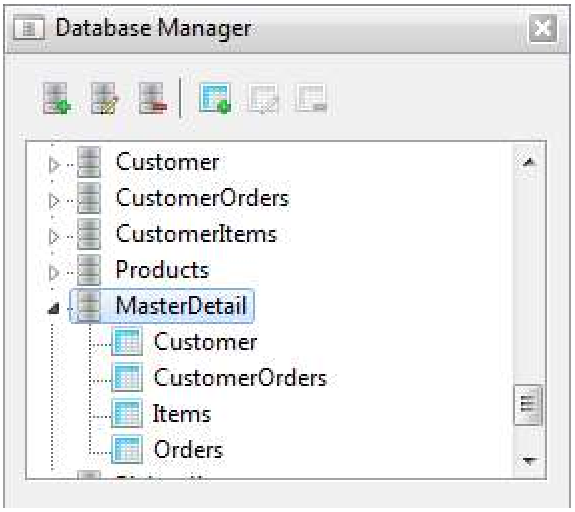
- Click on the Remove Database button in the database manager toolbar:

- A confirmation dialog will be displayed, asking you to confirm the removal of the database. Click on the Yes button to continue, or the No button to cancel the removal.

Adding a New DataSet

Use the following steps to add a new dataset using the database manager:

- Be sure that you have selected an existing database in the database manager by clicking on the desired existing database:



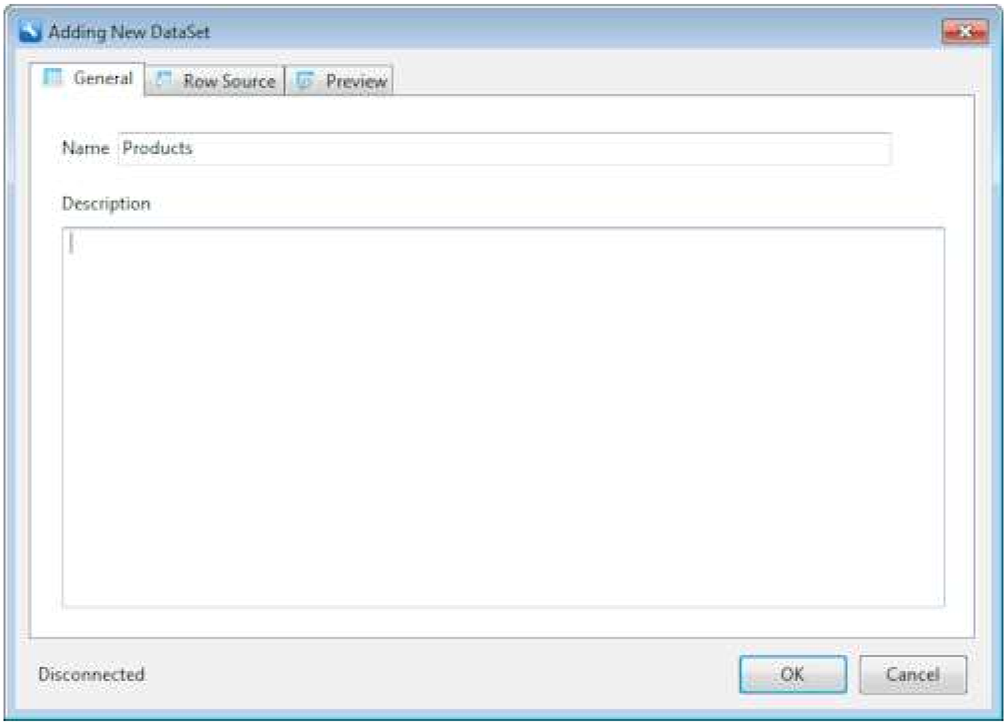
- Click on the Add DataSet button in the database manager toolbar:

- The dataset editor dialog will appear. Please refer to the next section for information on defining the dataset.

Defining a DataSet

The dataset editor dialog consists of 3 pages:

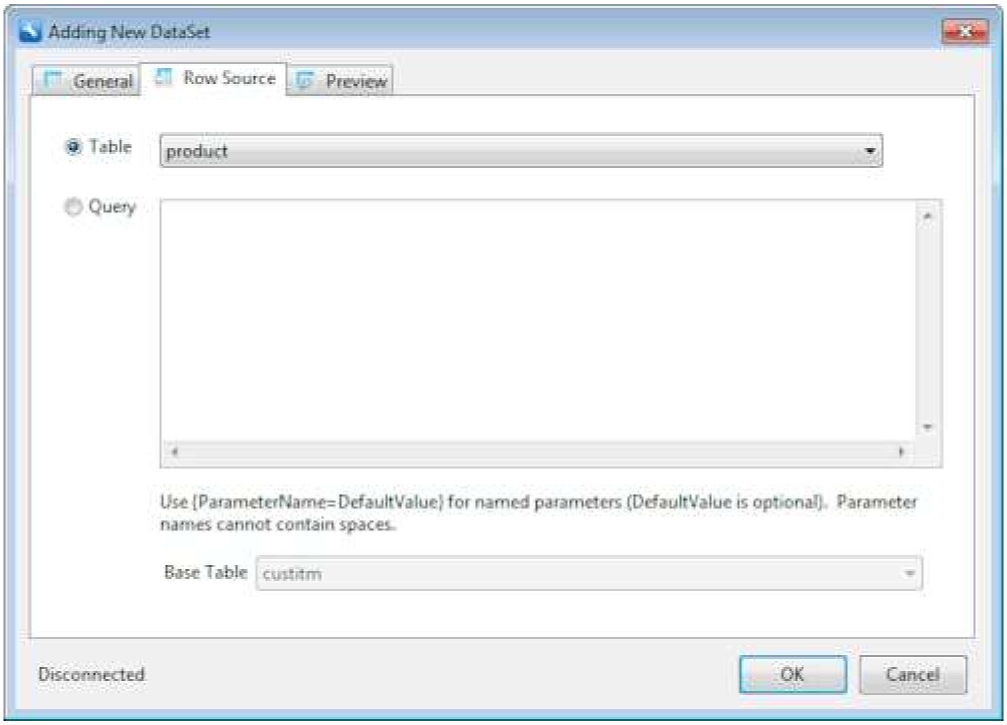
- **General** - the name of the dataset and the description.



Name Products

Description

- **Row Source** - the actual source of the dataset rows can be an actual table name from the selected database, or it can be an SQL SELECT statement.



● Table: product

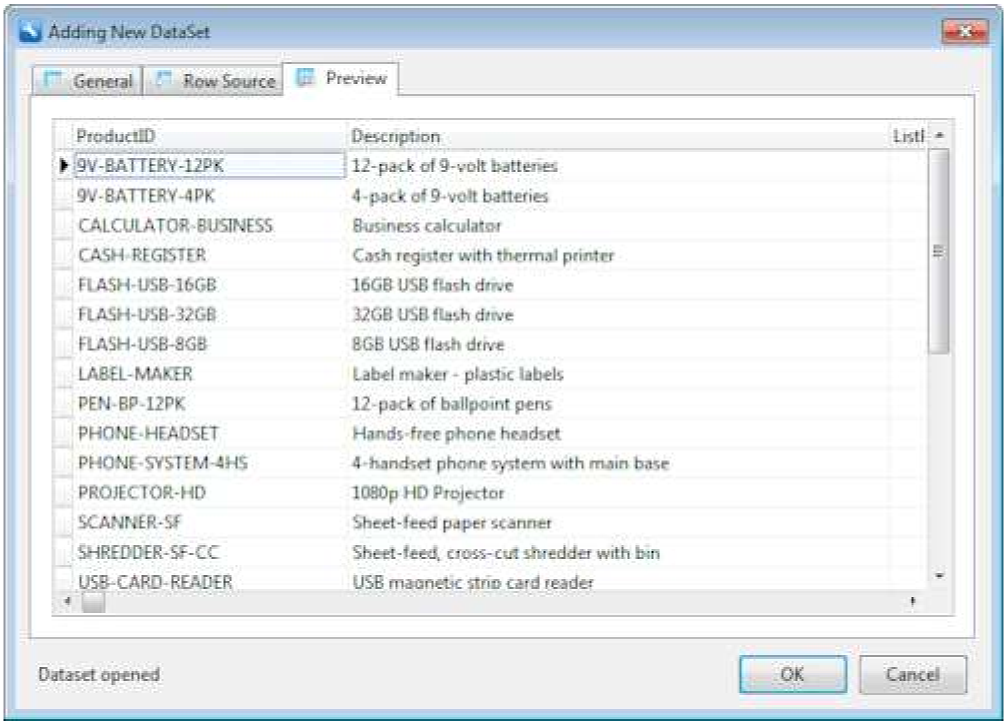
○ Query

Use {ParameterName=DefaultValue} for named parameters (DefaultValue is optional). Parameter names cannot contain spaces.

Base Table: custitm

Elevate Web Builder uses a special parameter naming syntax for queries, and does **not** use the native parameter functionality in the target database engine. This is done because some database engines do not support named parameters, or do not support parameter type discovery or enumeration. When the dataset rows are requested from the internal web server embedded in the IDE, it automatically populates the named parameters in the query by using the URL "name=value" parameters passed with the dataset rows request. These parameters can be specified in the application via the TDataSet Params property.

- **Preview** - use the preview page to make sure that the dataset is returning the correct rows. Any default values for parameters defined on the Row Source page are applied for the preview, so if you have not defined any default parameter values you may see zero rows displayed.



ProductID	Description	ListI
9V-BATTERY-12PK	12-pack of 9-volt batteries	
9V-BATTERY-4PK	4-pack of 9-volt batteries	
CALCULATOR-BUSINESS	Business calculator	
CASH-REGISTER	Cash register with thermal printer	
FLASH-USB-16GB	16GB USB flash drive	
FLASH-USB-32GB	32GB USB flash drive	
FLASH-USB-8GB	8GB USB flash drive	
LABEL-MAKER	Label maker - plastic labels	
PEN-BP-12PK	12-pack of ballpoint pens	
PHONE-HEADSET	Hands-free phone headset	
PHONE-SYSTEM-4HS	4-handset phone system with main base	
PROJECTOR-HD	1080p HD Projector	
SCANNER-SF	Sheet-feed paper scanner	
SHREDDER-SF-CC	Sheet-feed, cross-cut shredder with bin	
USB-CARD-READER	USB magnetic strip card reader	

Dataset opened

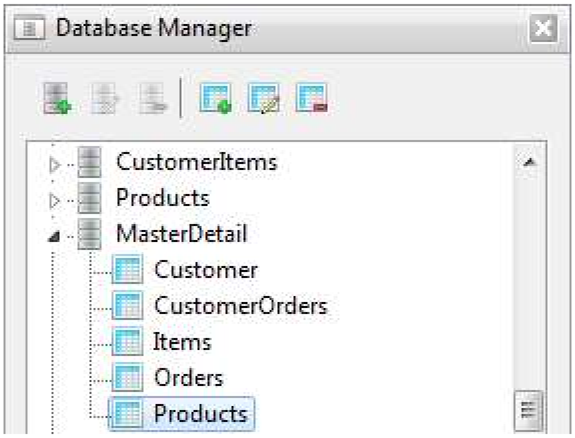
Editing an Existing DataSet

To edit an existing dataset using the database manager, simply double-click on the desired dataset in the list of datasets in the database manager. The dataset editor dialog will then appear, and you can use it to modify the dataset accordingly.

Removing a DataSet

Use the following steps to remove a dataset using the database manager:

- Click on the name of the dataset that you wish to remove:

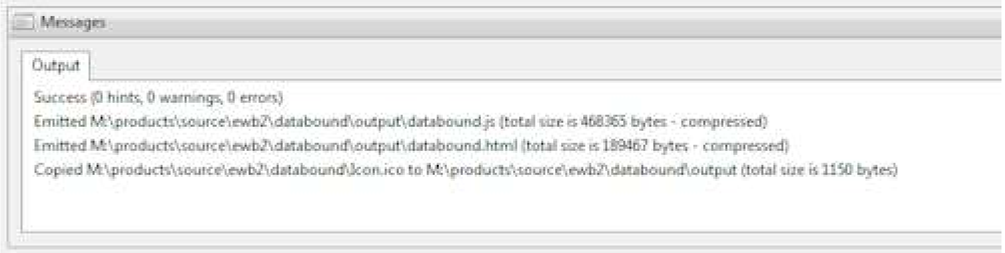


- Click on the Remove DataSet button in the database manager toolbar:

- A confirmation dialog will be displayed, asking you to confirm the removal of the dataset. Click on the Yes button to continue, or the No button to cancel the removal.

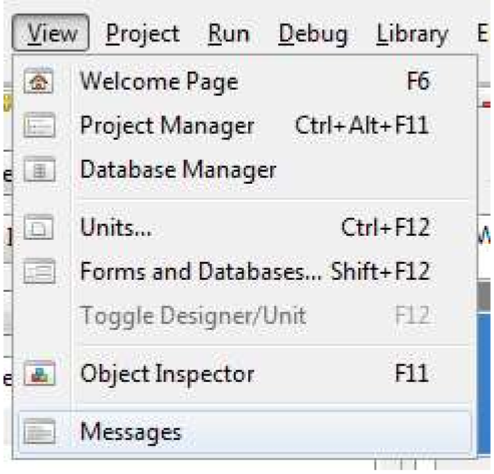
2.16 Viewing Messages

The messages view provides status information for compilation, as well as logging of debug messages sent from a web application. You can find out more information on sending debug messages to the IDE in the Debugging topic.



```
Success (0 hints, 0 warnings, 0 errors)
Emitted M:\products\source\ewb2\databound\output\databound.js (total size is 468365 bytes - compressed)
Emitted M:\products\source\ewb2\databound\output\databound.html (total size is 189467 bytes - compressed)
Copied M:\products\source\ewb2\databound\Icon.ico to M:\products\source\ewb2\databound\output (total size is 1150 bytes)
```

By default, the messages panel is visible in the IDE. If the messages panel is closed, you can open it by using the Messages option on the View menu:



Compilation Messages

There are three types of messages that may appear during compilation of an application:

Message Type	Description
Error	This message indicates that an error has occurred during compilation. You can double-click on the error to go to the source unit line responsible for the error. Compilation errors are fatal, and prevent the compiler from successfully emitting an application.
Warning	This message indicates that the compiler is warning that there is source code present that may cause run-time errors if not corrected. An example of this would be a reference to an uninitialized variable. You can double-click on the error to go to the source unit line responsible for the warning. Compilation warnings are not fatal, but one should always make sure to change the source code to remove such warnings in order to ensure that the compiled application is as reliable as possible.
Hint	This message indicates that the compiler has a hint regarding the compilation. An example of this would be a variable that is declared but never actually referenced. You can double-click on the hint to go to the source unit line responsible for the hint. Compilation hints are not fatal and can be safely ignored.

After the compilation of an application has successfully completed, or has failed, you will also see a status message summarizing the result of the compilation, including messages indicating which output files were emitted, and their location.

Deployment Messages

During deployment of an application, the copying of all application files is logged in the messages panel, one line per file.

Design-Time Execution Messages

In rare cases, component library code that contains one or more bugs may cause an exception at design-time. In such a case, you'll see a runtime error message appear in the messages panel. Double-clicking on the runtime error message will display a debug dialog that will show you the complete error message along with a call stack trace up until the point of the exception.

HTML Form Submittal Messages

If you use the special form submittal URL (http://localhost/formsubmit) to submit an HTML form, the results of the submittal will be echoed back to the messages panel. Double-clicking on the form submittal results will display a debug dialog that will show you the complete set of form values received by the web server. This works with both the internal web server and any external EWB Web Server. Please see the Using HTML Forms topic for more information.

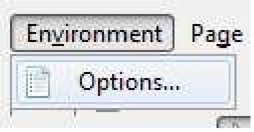
2.17 Modifying Environment Options

The Environment Options dialog allows you to configure the following aspects of the Elevate Web Builder IDE:

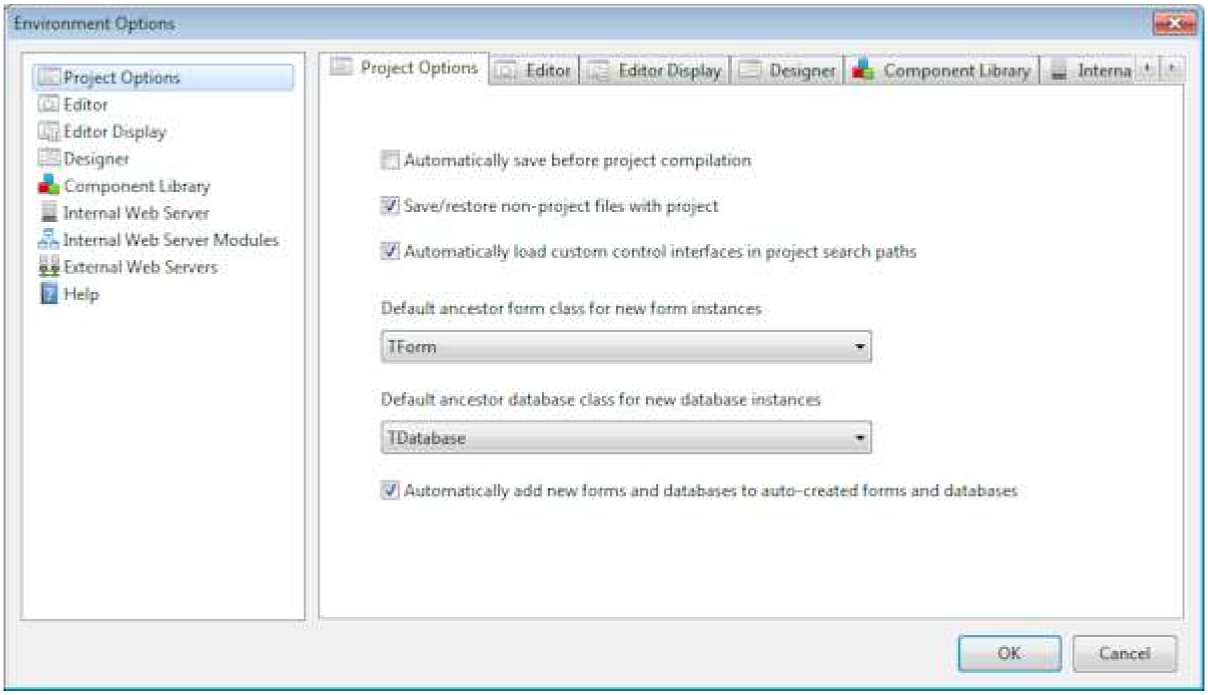
- The project options settings
- The code editor settings
- The code editor display settings
- The designer settings
- The component library settings
- The internal web server settings
- The web server modules added to the internal web server
- The external web servers added to the IDE
- The help files added to the IDE

Use the following steps to modify the environment options for the IDE:

- Click on the **Environment** option in the main menu. The Environment menu will open:



- Click on the **Options** option in the Environment menu to open the Environment Options dialog.



☐ Automatically save before project compilation

☑ Save/restore non-project files with project

☑ Automatically load custom control interfaces in project search paths

Default ancestor form class for new form instances

TForm

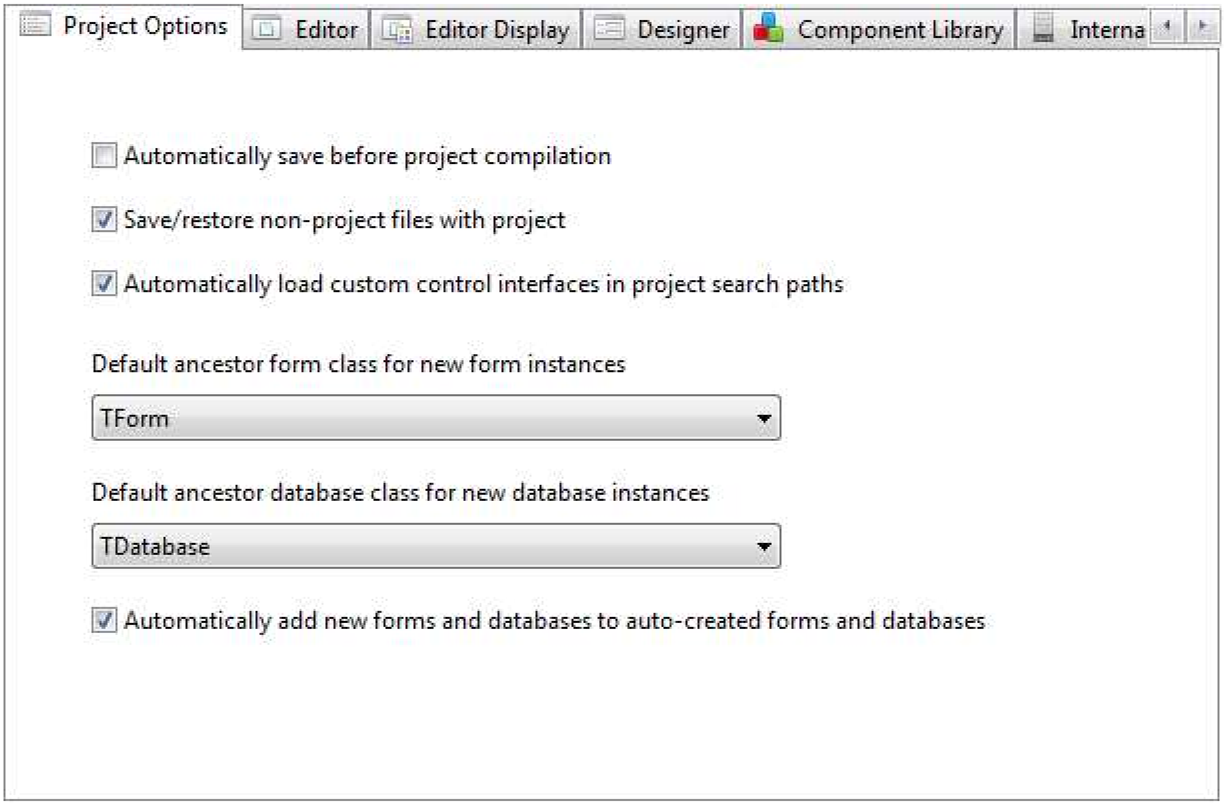
Default ancestor database class for new database instances

TDatabase

☑ Automatically add new forms and databases to auto-created forms and databases

Project Options

The Project Options page provides options for modifying the project options settings.



□ Automatically save before project compilation

☑ Save/restore non-project files with project

☑ Automatically load custom control interfaces in project search paths

Default ancestor form class for new form instances

TForm

Default ancestor database class for new database instances

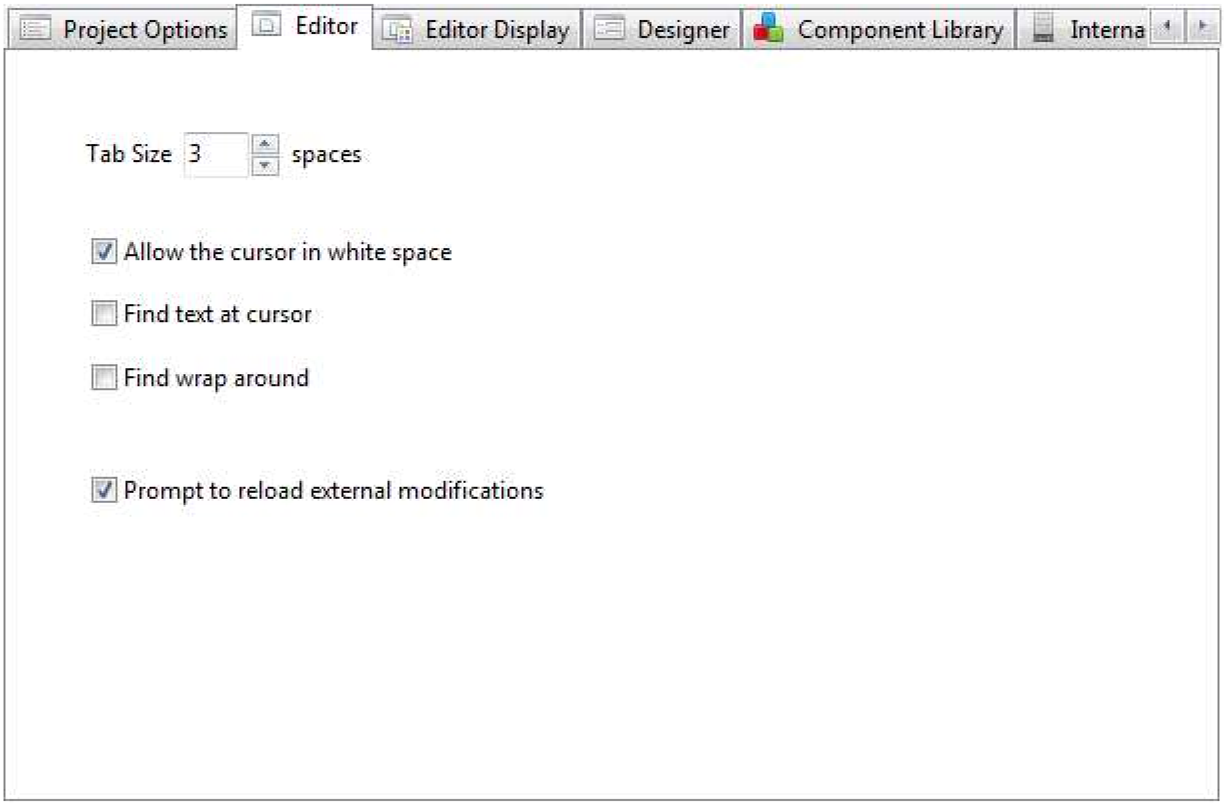
TDatabase

☑ Automatically add new forms and databases to auto-created forms and databases

Option	Description
Automatically save before project compilation	Select this check box to make sure that the IDE automatically saves all modified units and project files before compiling the currently-loaded project. This option is selected, by default.
Save/restore non-project files with project	Select this check box to have the IDE automatically save and restore any units that are open in the IDE, but are not actually part of the currently-loaded project. This option also applies to control interfaces that are open in the IDE and is selected, by default.
Automatically load custom control interfaces in project search paths	Select this check box to have the IDE automatically load any custom control interface files located in the project's compiler search paths whenever a project is opened in the IDE. When checking to see if a control interface has been customized, the IDE compares the path of the default control interface file used with the component library (based upon the Library Search Paths setting on the Component Library page) with the path of any control interfaces with the same file name present in the project's compiler search paths. If a match is found, then the control interface file found in the project's compiler search paths is loaded into the IDE and used with the project's form designers. After the project is closed, the default control interfaces are reloaded. This check box is selected, by default.
Default ancestor form class for new form instances	Specifies the default ancestor form class for the new form class selection dialog that is displayed when creating a new form in the IDE. The default ancestor form class is the TForm class.
Default ancestor database class for new database instances	Specifies the default ancestor database class for the new database class selection dialog that is displayed when creating a new database in the IDE. The default ancestor database class is the TDatabase class.
Automatically add new forms and databases to auto-created forms and databases	Select this check box to make sure that any newly-create forms and databases are automatically added to the list of auto-created forms and databases for the application. This option is selected, by default.

Editor

The Editor page provides options for modifying the code editor settings.



Tab Size 3 spaces

☑ Allow the cursor in white space

□ Find text at cursor

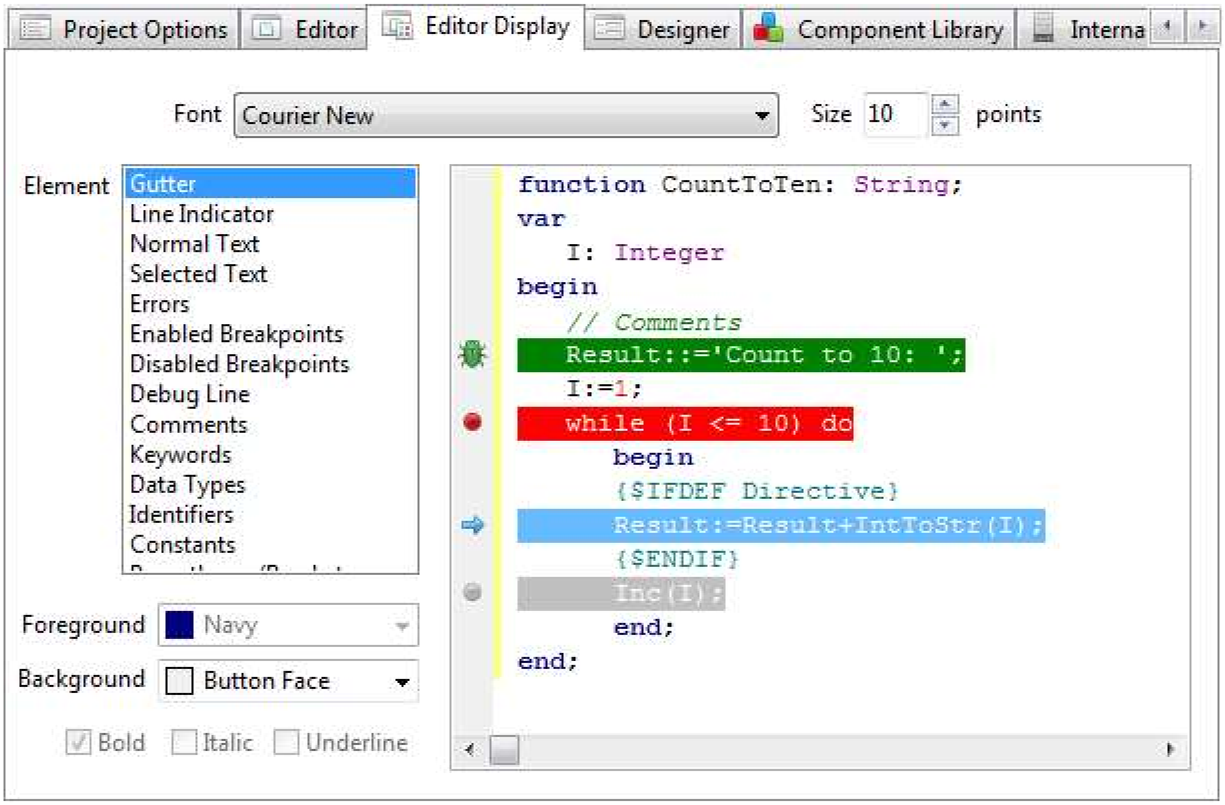
□ Find wrap around

☑ Prompt to reload external modifications

Option	Description
Tab Size	The number of spaces between each tab position. The default is 3 spaces.
Allow the cursor in white space	Select this check box in order to allow the cursor to be positioned in white space areas in the code editor. By default, if you move to an area of the code editor that is white space, the cursor will be moved to the next closest source code to the white space. The definition of "white space" in this context is the area of the code editor where there is no source code present.
Find text at cursor	Select this check box to have the code editor populate the Find or Replace search text box with the current word under the cursor when searching or replacing text in the code editor. By default, the last searched text will appear in the Find or Replace search text box.
Find wrap around	Select this check box to have the code editor wrap around to the start/end of the source when searching or replacing text in the code editor. The direction in which the searching or replacing wraps is determined by the direction of the search or replace operation. By default, the code editor will stop when reaching the start/end of the source during a search or replace operation.
Prompt to reload external modifications	Select this check box to have the code editor prompt the user when any source loaded in the code editor is modified by an external application. The prompt will ask the user to confirm whether they wish to load the modified source into the code editor. By default, the code editor will prompt the user when any source is changed by an external application.

Editor Display

The Editor Display page provides options for modifying the code editor settings.



Font Courier New

Size 10 points

Element
- Gutter
- Line Indicator
- Normal Text
- Selected Text
- Errors
- Enabled Breakpoints
- Disabled Breakpoints
- Debug Line
- Comments
- Keywords
- Data Types
- Identifiers
- Constants

Foreground 🟦 Navy

Background □ Button Face

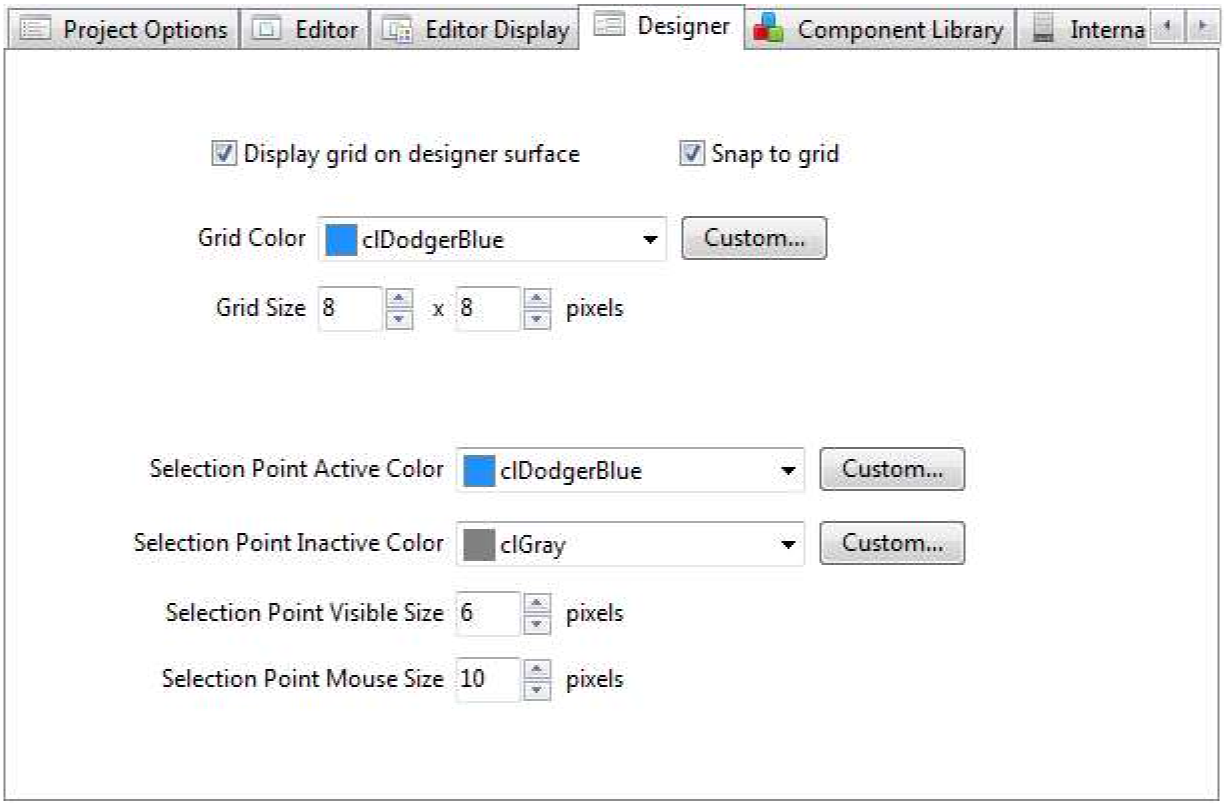
☑ Bold □ Italic □ Underline

```
function CountToTen: String;
var
  I: Integer
begin
  // Comments
  Result::='Count to 10: ';
  I:=1;
  while (I <= 10) do
    begin
    {$IFDEF Directive}
    Result:=Result+IntToStr(I);
    {$ENDIF}
    Inc(I);
    end;
end;
```

Option	Description
Font	Use this combo box to select the fixed-width font to use for all text in the code editor. The default code editor font is the "Courier New" font.
Size	The size of the fixed-width font, in points. The default size is 10 points.
Element	Use this list box to select the various text elements present in the code editor and modify their visual properties such as their foreground and background colors and the style of the text.

Designer

The Designer page provides options for modifying the designer settings.



☑ Display grid on designer surface ☑ Snap to grid

Grid Color 🟦 clDodgerBlue

Grid Size 8 x 8 pixels

Selection Point Active Color 🟦 clDodgerBlue

Selection Point Inactive Color ⬛ clGray

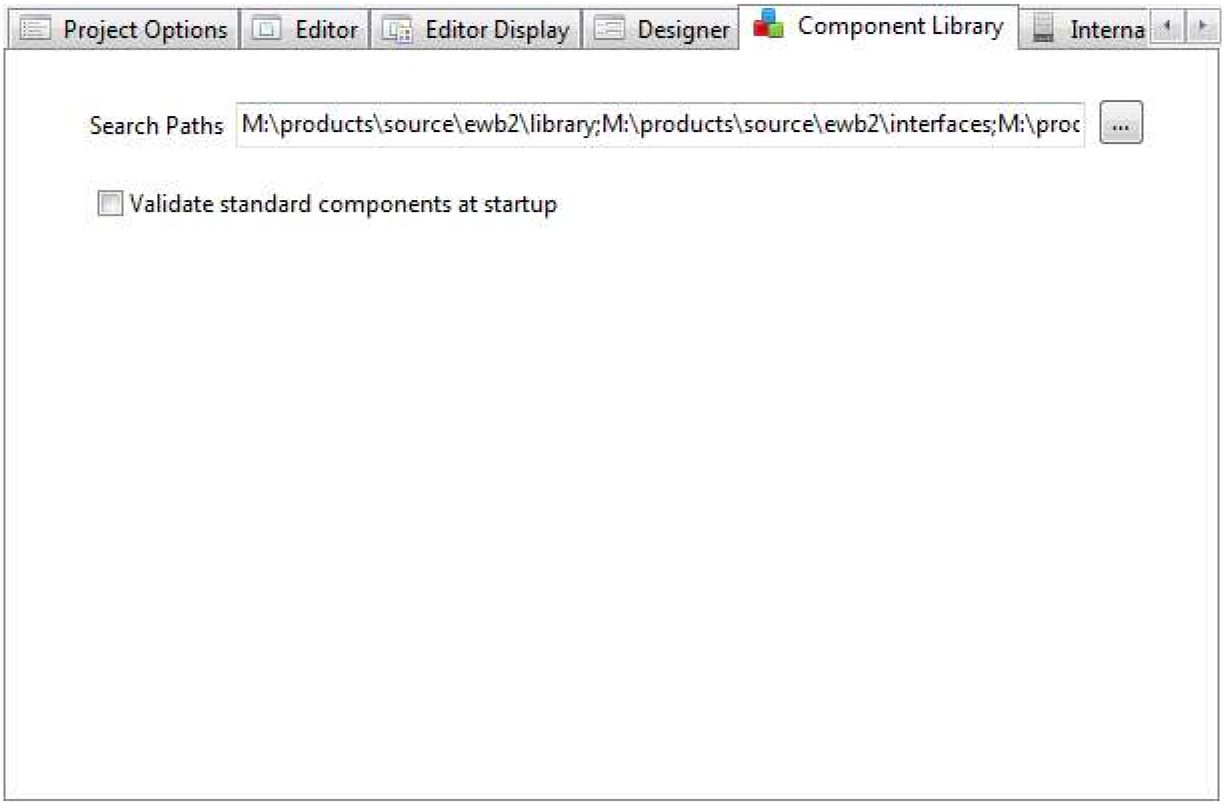
Selection Point Visible Size 6 pixels

Selection Point Mouse Size 10 pixels

Option	Description
Display grid on designer surface	Select this check box to enable the display of an alignment grid on the designer surface. The default state is checked.
Snap controls to grid	Select this check box to cause the designer to automatically align any controls/elements to the grid when they are inserted, resized, or moved. The default state is checked.
Grid Color	Select the color of the alignment grid. The default is clDodgerBlue.
Grid Size	The number of pixels between each grid point in the alignment grid, both on the horizontal (X) and vertical (Y) axes. The default grid size is 8 pixels by 8 pixels.
Selection Point Active Color	Select the color of selection points when the designer is active. The default is clDodgerBlue.
Selection Point Inactive Color	Select the color of selection points when the designer is not active. The default is clGray.
Selection Point Visible Size	Use this edit to specify the visible size of selection points. The default size is 6 pixels (square).
Selection Point Mouse Size	Use this edit to specify the size of the area in which the mouse can operate on the selection points. If you are visually-impaired, then you may want to increase these values to make working with the selection points easier. The default is 10 pixels (square).

Component Library

The Component Library page provides options for modifying the component library settings.



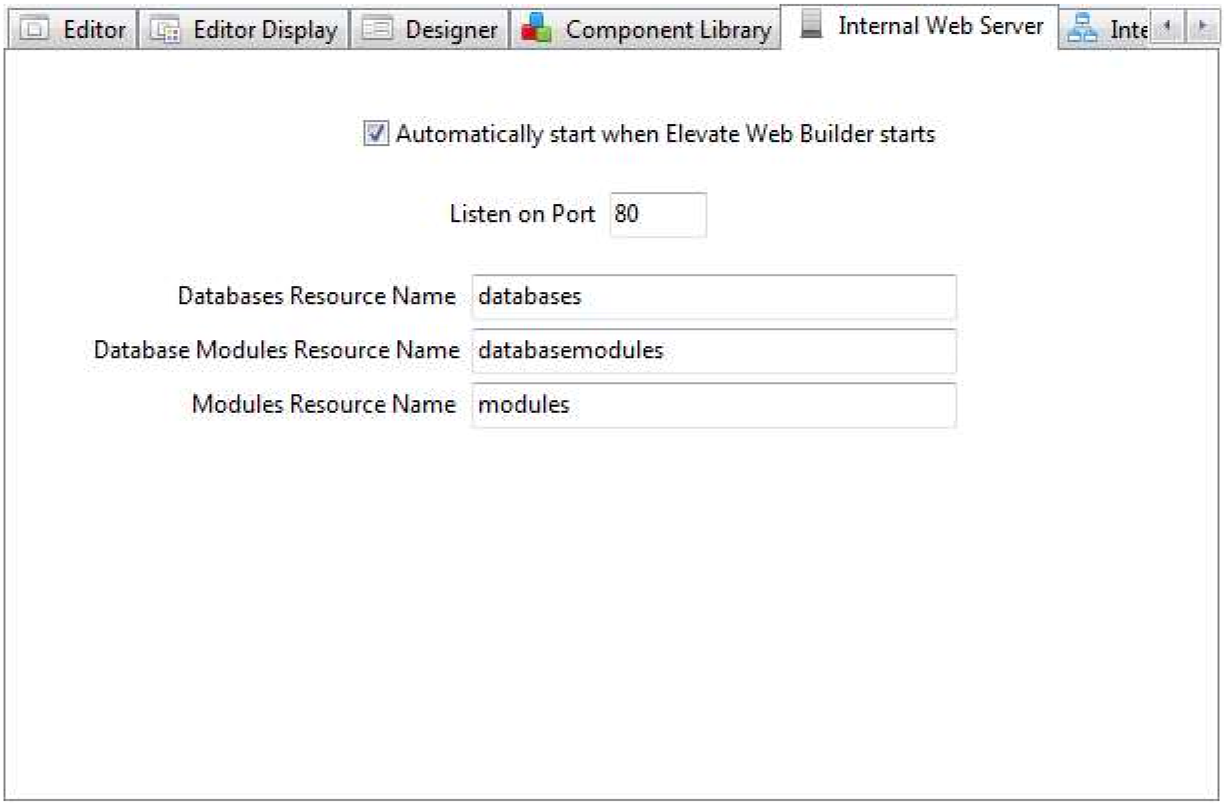
Search Paths M:\products\source\ewb2\library;M:\products\source\ewb2\interfaces;M:\proc

□ Validate standard components at startup

Option	Description
Search Paths	The component library search paths are used to specify where the component library source unit files are located. These search paths ensure that the compiler can always find the component units, and any referenced control interfaces, installed into the component library, as well as any core units that are necessary for all Elevate Web Builder applications. The component library search paths are initially configured during installation. If you wish to add additional paths to the component library search paths, then this is where you would do so. When specifying more than one search path, be sure to separate multiple paths with a semicolon (;). **Note** These search paths are global to both applications and the component library, but the project's search paths always take precedence over these search paths.
Validate standard components at startup	Select this check box to have the IDE check for the existence of the standard Elevate Web Builder components during startup. If any of the standard components are missing, or not found in their default location, then the user will be asked to confirm adding the missing standard components. By default, the IDE will always validate the standard components during startup.

Internal Web Server

The Internal Web Server page provides options for modifying the internal web server settings.



☑ Automatically start when Elevate Web Builder starts

Listen on Port: 80

Databases Resource Name: databases

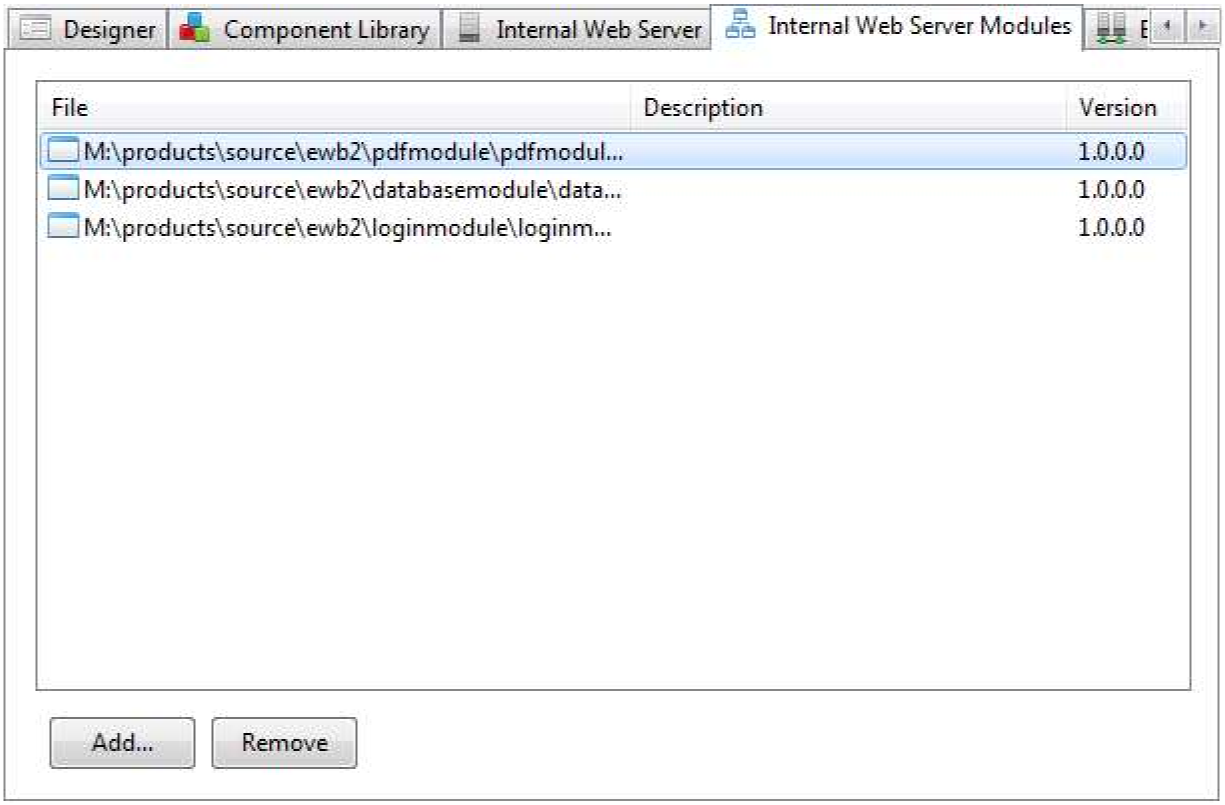
Database Modules Resource Name: databasemodules

Modules Resource Name: modules

Option	Description
Auto-Start	Select this check box to specify that the internal web server should be automatically started when the IDE is started. The default is checked.
Listen on Port	Use this edit to specify the port number that the internal web server should listen on. The default is port 80.
Databases Resource Name	Specifies the resource name to use for the automatic database handling built into the internal web server. The default value is 'databases'. Please see the Creating and Loading DataSets topic for more information on how this resource name is used in database requests.
Database Modules Resource Name	Specifies the resource name to use for any database modulesadded to the internal web server (see next). The default value is 'databasemodules'. Please see the Creating and Loading DataSets topic for more information on how this resource name is used in database requests.
Modules Resource Name	Specifies the resource name to use for the modules added to the internal web server. The default value is 'modules'. Please see the Creating Web Server Modules topic for more information on how this resource name is used in module requests.

Internal Web Server Modules

The Internal Web Server Modules page provides options for adding and removing modules (*.dll) that were created using Embarcadero RAD Studio and Delphi and an Elevate Web Builder Module template project from the repository in the RAD Studio IDE. Adding modules to the internal web server allows the modules to be used to respond to requests and provide content to the Elevate Web Builder application running in the IDE.



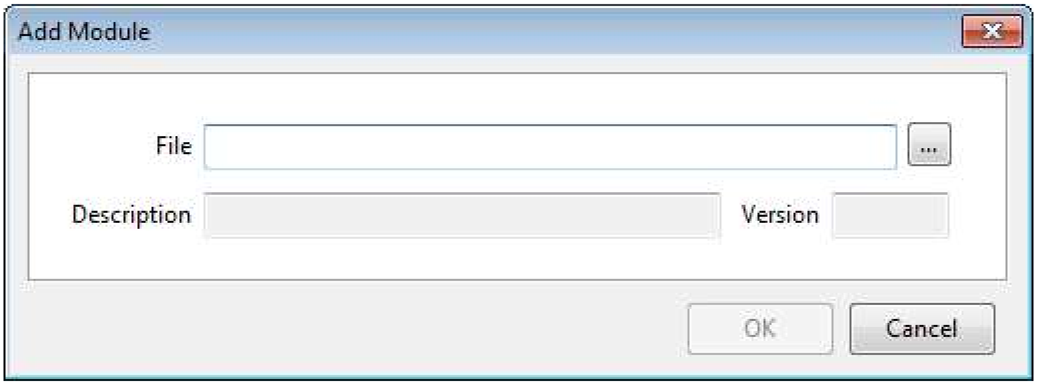
File	Description	Version
M:\products\source\ewb2\pdfmodule\pdfmodul...		1.0.0.0
M:\products\source\ewb2\databasemodule\data...		1.0.0.0
M:\products\source\ewb2\loginmodule\loginm...		1.0.0.0

Adding a Module

In order to add a module, complete the following steps:

- Click on the Add button

- The Add Module dialog will appear.



In the dialog, specify the file name of the module (.dll) that you wish to add to the internal web server. You can type in the file name directly, or use the browse button (...) to select the module using a common Windows file dialog. If you use the browse button, the module description and version will be populated from the module after the file is selected. The description and version are read directly from the .dll's version information.

- Click on the OK button. If the specified file is a valid Elevate Web Builder module, then the module will be added to the internal web server. If the specified file is not a valid module file, then an error message will be displayed indicating any issues with the module file.

Removing a Module

In order to remove a module, complete the following steps:

- Select an existing module from the list of modules.

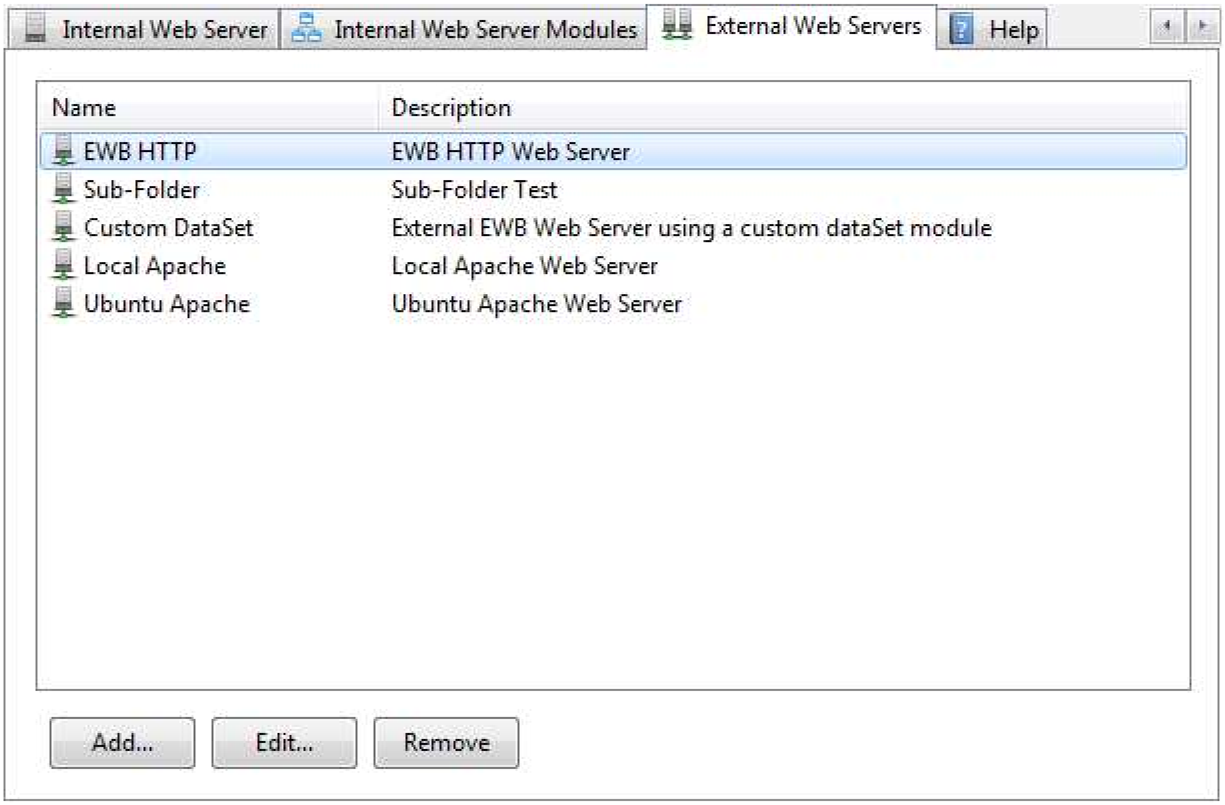
- Click on the Remove button.

> **Note**
> If you remove a module that is used by Elevate Web Builder applications, then you will experience errors in these applications when they try to execute requests that reference these modules in the URL for the request.

Please see the Creating Web Server Modules topic for more information how the modules work.

External Web Servers

The External Web Servers page provides options for adding external web servers for use in the IDE. Once an external web server is added, it can be selected as the target web server when running applications. Please see the Running a Project topic for more information on running applications.

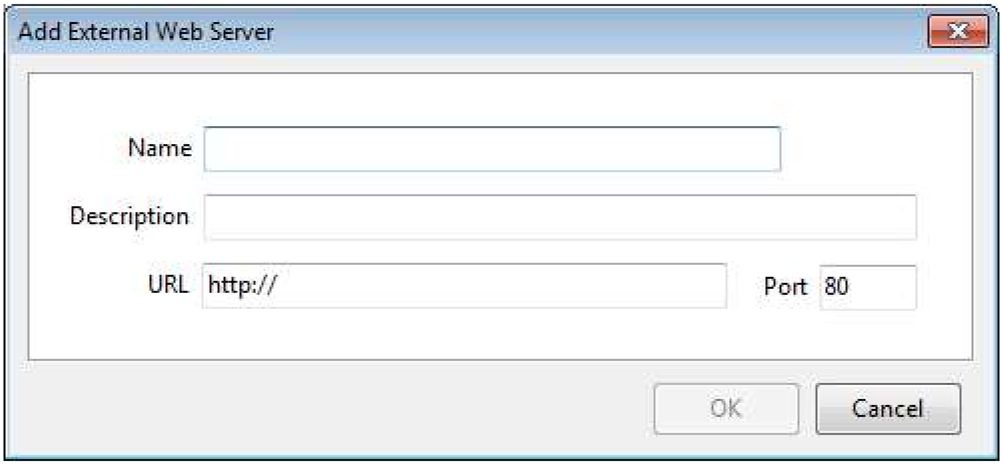


Name	Description
EWB HTTP	EWB HTTP Web Server
Sub-Folder	Sub-Folder Test
Custom DataSet	External EWB Web Server using a custom dataSet module
Local Apache	Local Apache Web Server
Ubuntu Apache	Ubuntu Apache Web Server

Adding an External Web Server

In order to add an external web server, complete the following steps:

- Click on the Add button.
- The Add External Web Server dialog will appear.



URL http://

Port 80

- In this dialog, specify the name of the external web server that you wish to add to the IDE. This will be used to uniquely identify the external web server.

  Next, specify the short description of the external web server. This will be used in the web server selection combo box in the IDE.

  Next, specify the URL of the external web server. This will be used by the web browser in the IDE to load an application from the external web server when is it is the currently-selected web server.

  Finally, specify the port on which the external web server will listen for requests from the web browser in the IDE. The default port is 80, which is the standard web server port (HTTP protocol).

- Click on the OK button. If all information for the external web server is specified correctly, then the external web server will be added to the IDE for use with your projects. If the specified external web server information is missing or invalid, then an error message will be displayed indicating any issues with the information.

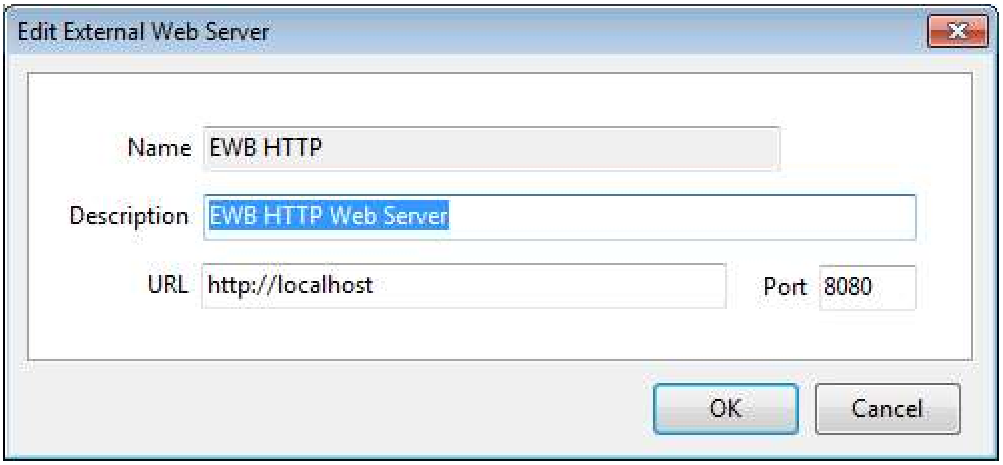
Editing an External Web Server

In order to edit an external web server that is already added, complete the following steps:

- Select an existing external web server from the list of web servers.

- Click on the Edit button.

- The Edit External Web Server dialog will appear.



Edit External Web Server

Name: EWB HTTP

Description: EWB HTTP Web Server

URL: http://localhost

Port: 8080

- Modify the external web server information as required.

- Click on the OK button. If all information for the external web server is specified correctly, then the external web server will be added to the IDE for use with your projects. If the specified external web server information is missing or invalid, then an error message will be displayed indicating any issues with the information.

Removing an External Web Server

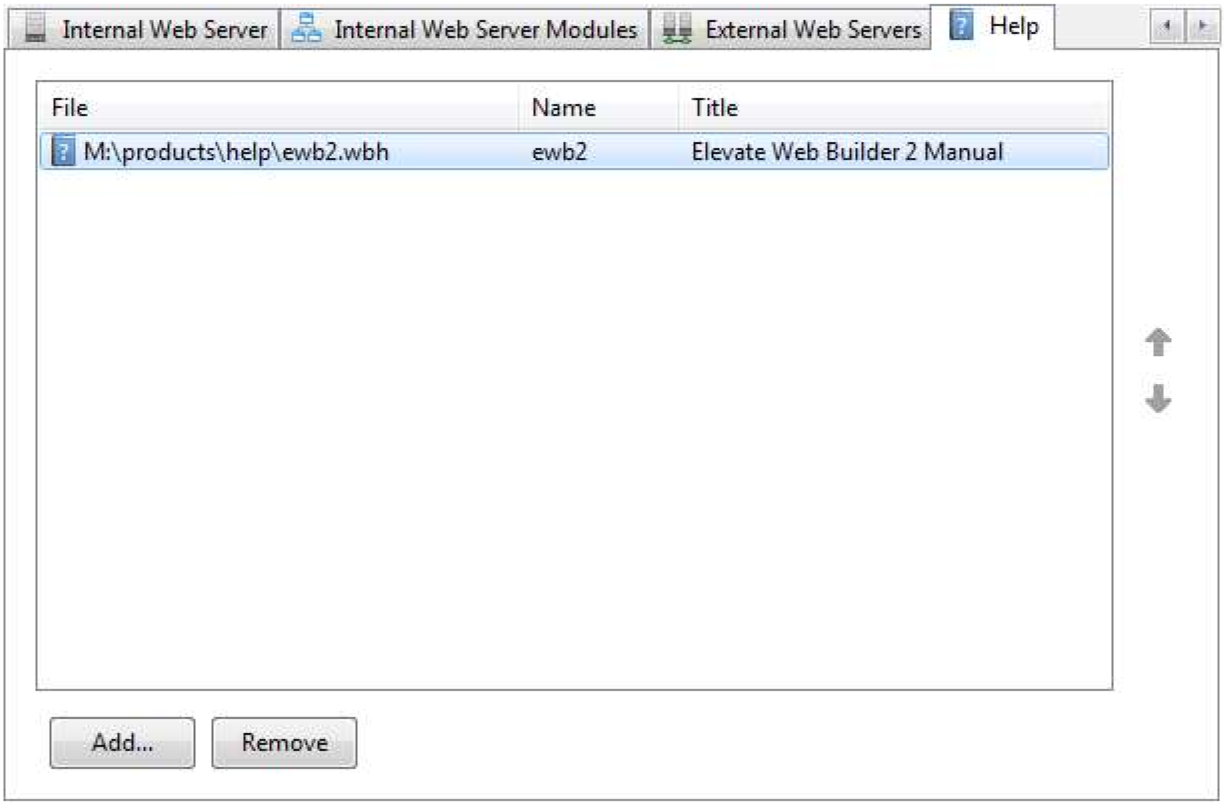
In order to remove an external web server, complete the following steps:

- Select an existing external web server from the list of web servers.
- Click on the Remove button.

Help

The Help page provides options for adding and removing help files (*.wbh). By default, the help for Elevate Web Builder is added automatically during the IDE startup process, so normally you will not need to add any additional help files. However, if you install any 3rd party components into the IDE, they may come with online help to use with the components, and that help can be added here.

> **Note**
> The default Elevate Web Builder help file is always shown in the list of added help files, but it cannot be removed.



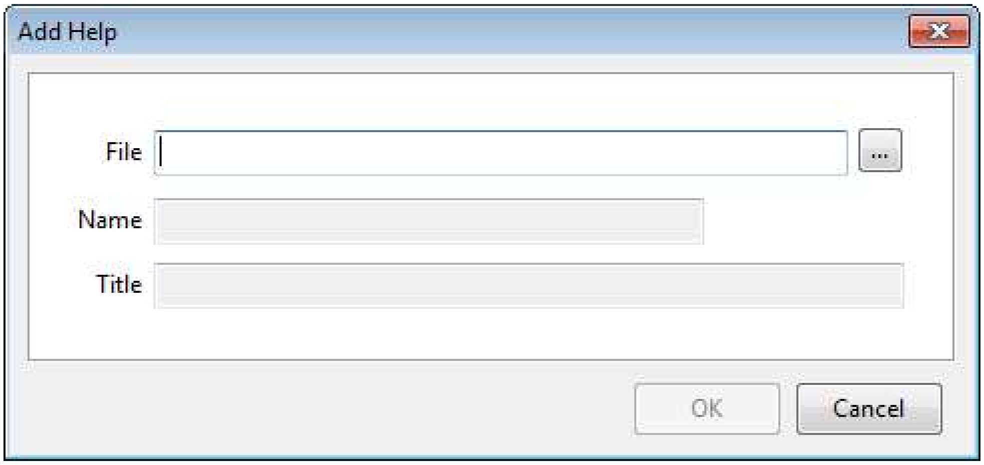
File	Name	Title
M:\products\help\ewb2.wbh	ewb2	Elevate Web Builder 2 Manual

Adding Help

In order to add a help, complete the following steps:

- Click on the Add button.

- The Add Help dialog will appear.



- In this dialog, specify the file name of the help file (.wbh) that you wish to add to the IDE in the edit control. You can type in the file name directly, or use the browse button (...) to select the help file using a common Windows file dialog. If you use the browse button, the help file name and title will be populated from the help file after the file is selected.

- Click on the OK button. If the specified file is a valid Elevate Web Builder help file, then the help file will be added to the IDE for use from the Help menu. If the specified file is not a valid help file, then an error message will be displayed indicating any issues with the help file.

Removing Help

In order to remove a help file, complete the following steps:

- Select an existing help file from the list of help files.

- Click on the Remove button.

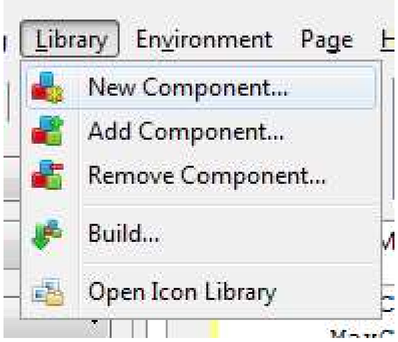
Please see the Accessing Help topic for more information on accessing the help in the IDE.

2.18 Creating a New Component

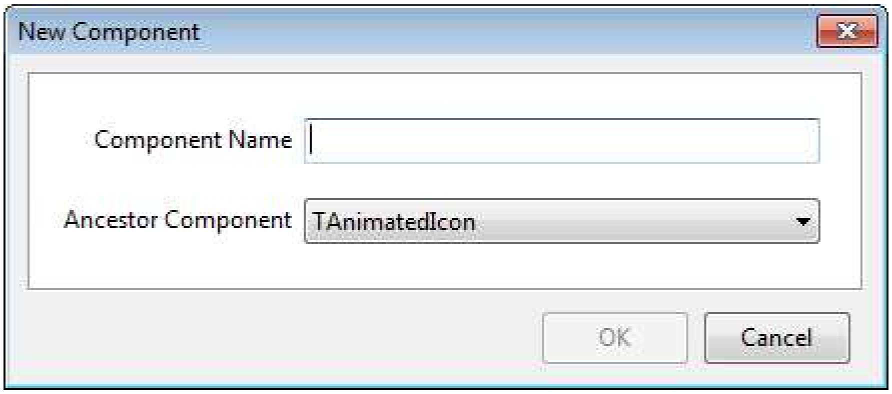
Use the following steps to create a new component in the IDE:

- Click on the **Library** option in the main menu. The Library menu will open.

- Click on the **New Component** option in the Library menu:



- The New Component dialog will now appear:



- In this dialog, specify the class name of the component that you wish to create in the first edit control. By convention, any class name in Elevate Web Builder should be prefixed with a capital "T".

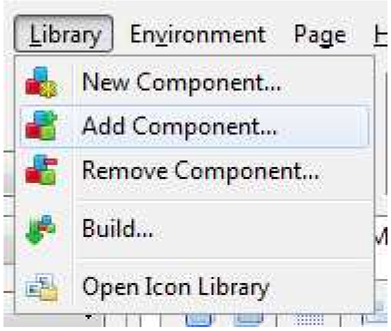
  Next, select the ancestor component class name for the new component by using the combo box provided.

- Click on the OK button. A new source unit containing the skeleton code for the new component class will now appear in the code editor. Please see the Using the Code Editor topic for more information on using the code editor.

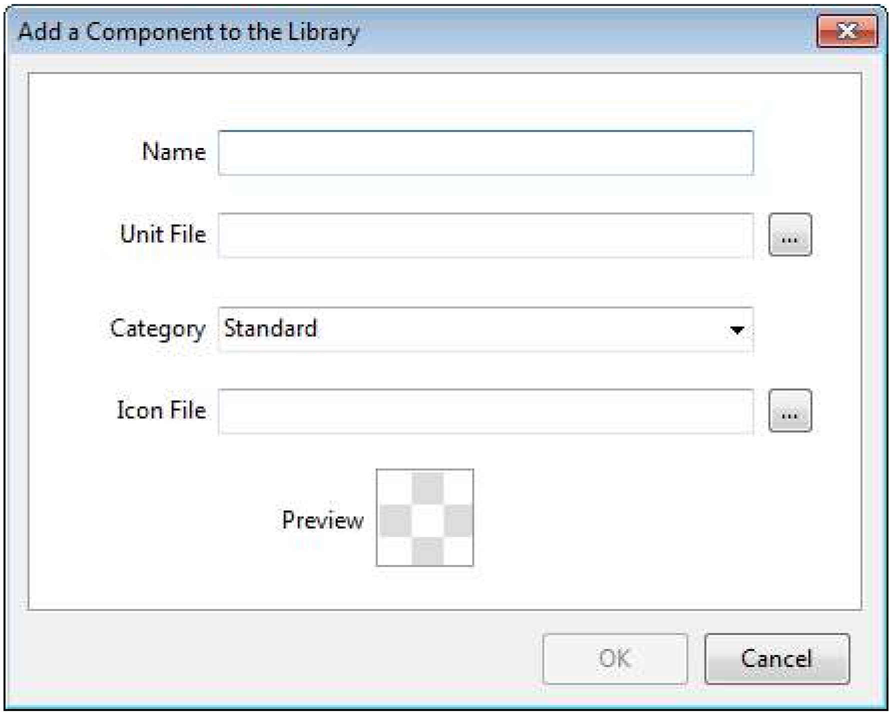
2.19 Adding a Component to the Component Library

Use the following steps to add a new component to the component library:

- Click on the **Library** option in the main menu. The Library menu will open.
- Click on the **Add Component** option in the Library menu:



- The Add Component dialog will now appear:



- In this dialog, specify the class name of the component that you wish to add to the component library in the first edit control.

  Next, specify the source unit file where the class name is declared. At least one of the declared classes in the source unit should match the specified class name. If not, an error will occur when the component library is rebuilt. You can type in the file name directly, or use the browse button (...) to select the source unit using a common Windows file dialog.

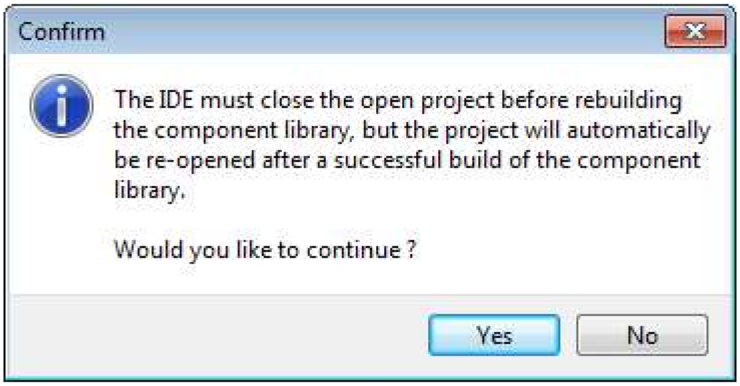
  Next, select an existing component palette category in which to place the component, or type in a new category in which to place the component using the combo box provided.

  Next, select an icon file to use to represent the component on the component palette. The icon file should be a 16 by 16 pixel PNG, JPEG, or GIF image file. You can type in the file name directly, or use the browse button (...) to select the icon file using a common Windows file dialog. After a valid file name has been specified or selected, a preview of the icon file will be shown in the Preview area.

> **Note**
> This step is optional, if you don't specify an icon file, or if the specified icon file is invalid, Elevate Web Builder will use a default, generic icon for the component on the component palette.

- Click on the OK button.

- If there is a project open in the IDE, then you will see the following dialog appear:



The IDE must close the open project before rebuilding the component library, but the project will automatically be re-opened after a successful build of the component library.

Would you like to continue ?

  Click on the Yes button to continue with rebuilding the component library.

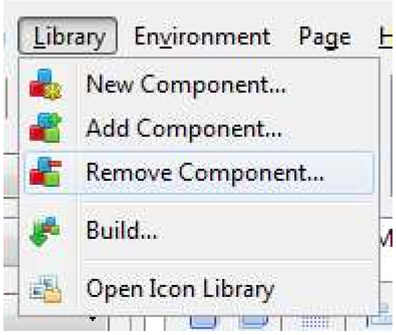
- The component library will now automatically be rebuilt and, if there were no errors during the compilation of the component library, the component just added will now appear on the component palette in the specified category. If there were one or more errors during the compilation of the component library, then you should correct the error(s) in the applicable source unit(s), and rebuild the component library manually. To see how to manually rebuild the component library, please see the Rebuilding the Component Library topic.

2.20 Removing a Component from the Component Library

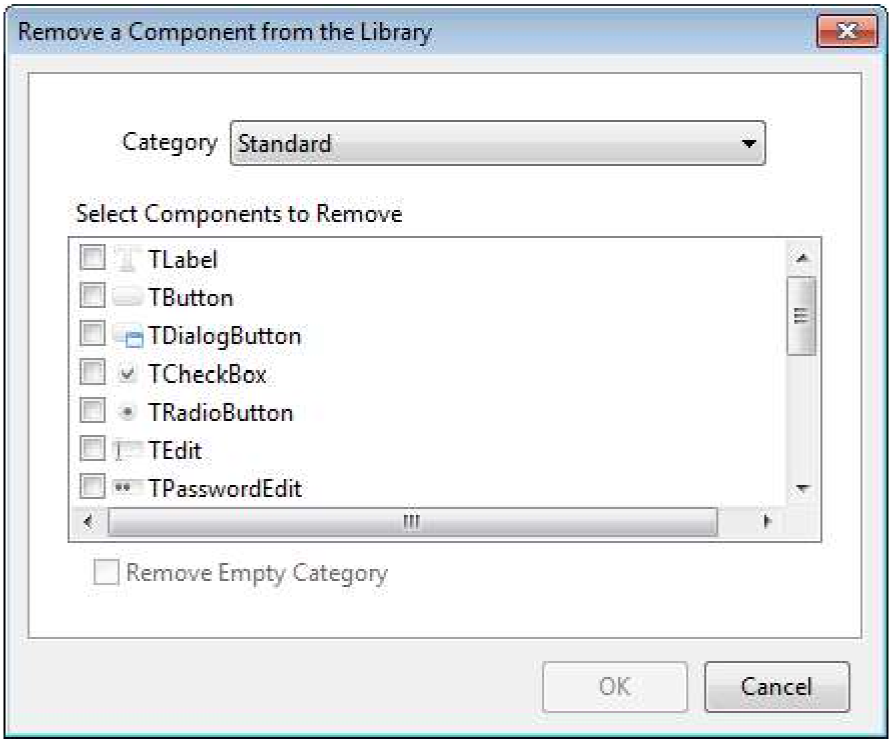
Use the following steps to remove an existing component from the component library:

- Click on the **Library** option in the main menu. The Library menu will open.

- Click on the **Remove Component** option in the Library menu:



- The Remove Component dialog will now appear:


Remove a Component from the Library

Category: Standard

Select Components to Remove

□ TLabel
□ TButton
□ TDialogButton
□ TCheckBox
□ TRadioButton
□ TEdit
□ TPasswordEdit

□ Remove Empty Category

- In this dialog, select the category where the component that you wish to remove is installed.

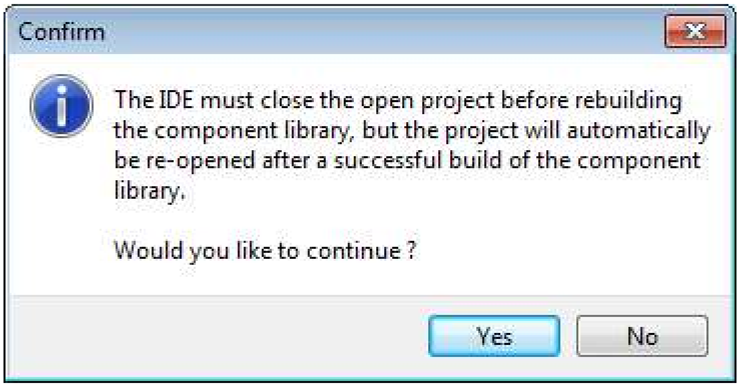
  Next, click on the check box next to any or all component(s) that you wish to remove from the component library.

  Next, click on the Remove Empty Category check box in order to also remove the selected category.

> **Note**
> The Remove Empty Category check box is only enabled if the selected category will be empty after removing the selected component(s).

- Click on the OK button.

- If there is a project open in the IDE, then you will see the following dialog appear:



The IDE must close the open project before rebuilding the component library, but the project will automatically be re-opened after a successful build of the component library.

Would you like to continue ?

  Click on the Yes button to continue with rebuilding the component library.

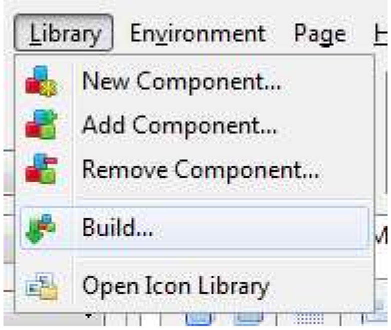
- The component library will now automatically be rebuilt and, if there were no errors during the compilation of the component library, the selected component(s) will be removed from the selected category on the component palette. If there were one or more errors during the compilation of the component library, then you should correct the error(s) in the applicable source unit(s), and rebuild the component library manually. To see how to manually rebuild the component library, please see the Rebuilding the Component Library topic.

> **Note**
> You should not normally encounter compilation errors when removing components from the component library. However, it is possible that one or more source units used in the component library have been modified since the last time the component library was rebuilt, and those modifications may have introduced compilation errors.

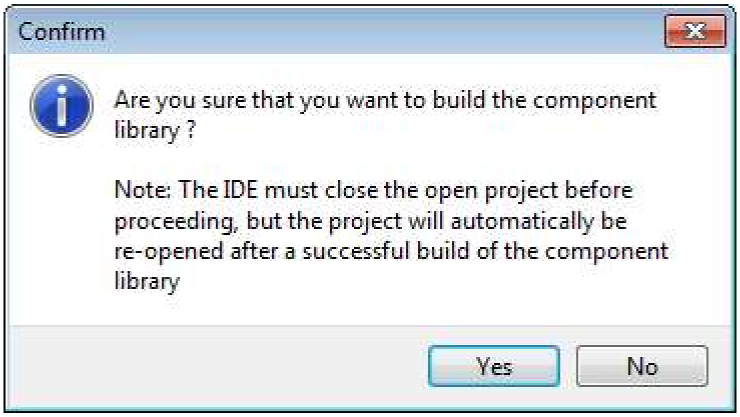
2.21 Rebuilding the Component Library

Use the following steps to rebuild the component library:

- Click on the **Library** option in the main menu. The Library menu will open.
- Click on the **Build** option in the Library menu:



- The following dialog will now appear:


Confirm

Are you sure that you want to build the component library ?

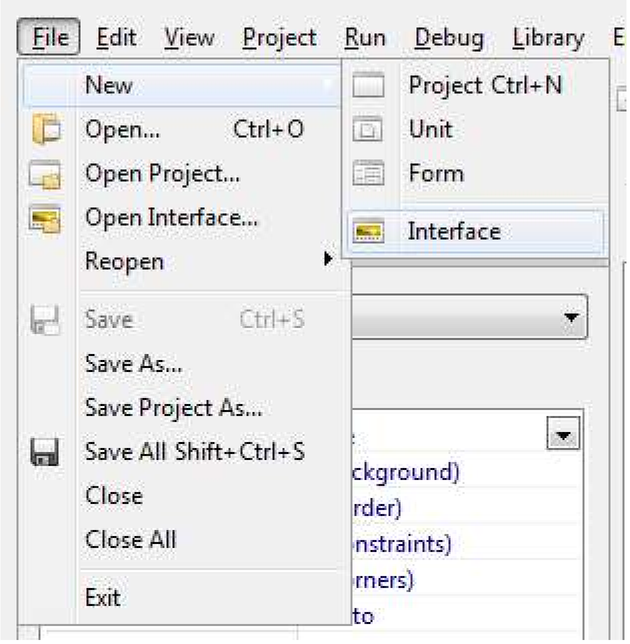
Note: The IDE must close the open project before proceeding, but the project will automatically be re-opened after a successful build of the component library

- Click on the Yes button to continue with rebuilding the component library.
- The component library will now automatically be rebuilt and, if there were no errors during the compilation of the component library, any changes to the source units and/or control interfaces used in the component library will be reflected in any open form and database designers. If there were one or more errors during the compilation of the component library, then you should correct the error(s) in the applicable source unit(s), and rebuild the component library again.

2.22 Creating a New Control Interface

Use the following steps to create a new control interface in the IDE:

- Click on the **File** option in the main menu. The File menu will open:



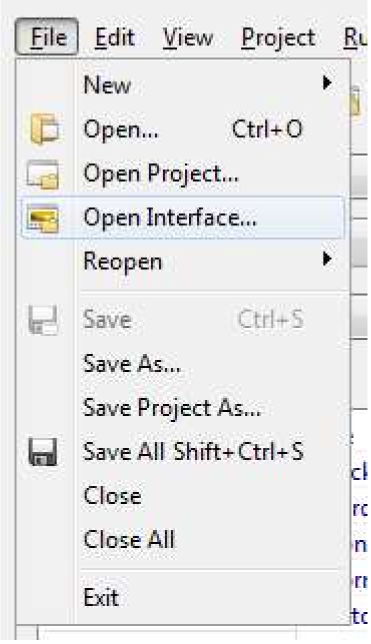
- Click on the **New** option in the File menu to open the New sub-menu. From the New sub-menu, select the **Interface** option.

- A new control interface will now appear in the Control Interface Editor.

2.23 Modifying a Control Interface

Use the following steps to modify an existing control interface in the IDE:

- Click on the **File** option in the main menu. The File menu will open:

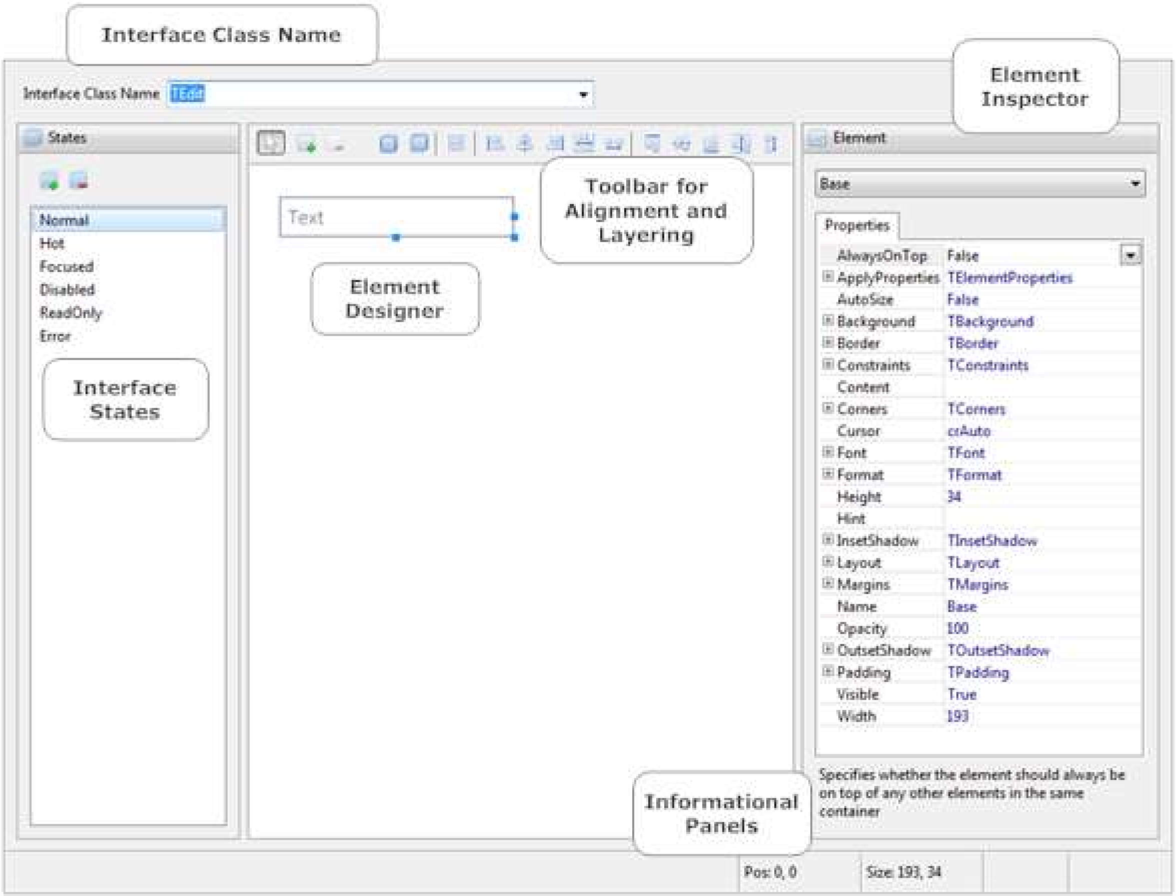


- Click on the **Open Interface** option in the File menu. A Windows file open dialog will appear. Navigate to, and select, the existing control interface that you wish to modify. Click on the Open button in the Windows file open dialog to complete opening the control interface.

- The existing control interface will now appear in the Control Interface Editor.

2.24 Using the Control Interface Editor

The control interface editor is used for creating new control interfaces or editing existing control interfaces. It has the following layout:



Interface Class Name TEdit

Base

AlwaysOnTop	False
⊞ ApplyProperties	TElementProperties
AutoSize	False
⊞ Background	TBackground
⊞ Border	TBorder
⊞ Constraints	TConstraints
Content	
⊞ Corners	TCorners
Cursor	crAuto
⊞ Font	TFont
⊞ Format	TFormat
Height	34
Hint	
⊞ InsetShadow	TInsetShadow
⊞ Layout	TLayout
⊞ Margins	TMargins
Name	Base
Opacity	100
⊞ OutsetShadow	TOutsetShadow
⊞ Padding	TPadding
Visible	True
Width	193

Specifies whether the element should always be on top of any other elements in the same container

Pos: 0, 0

Size: 193, 34

The unit of measure used by the interface designer is the pixel, and the resolution is always assumed to be 96 pixels per inch.

By default, the interface designer shows a grid to aid with component placement and alignment, and the grid guides (dots) are spaced apart at 8 pixel intervals. Please see the Modifying Environment Options topic for more information on modifying the interface designer grid properties.

Note
If you haven't already, please make sure to read the Control Interfaces topic before proceeding. It explains the structure of control interface and many of the control interface concepts that are used in the control interface editor.

Specifying the Interface Class Name

Use the Interface Class Name combo box to select an existing control class name for a control included in the component library, or type in a new interface class name. The interface class name normally corresponds to an existing control class name, but does not **always** do so. However, as discussed in the Control Interfaces topic, the specified interface class name **should** correspond to a value returned by the protected TControl GetInterfaceClassName method for one or more controls in the component library.

> **Note**
> If the interface class name does not correspond to any interface class names used by any controls in the component library, then the interface will effectively be ignored by the component library.

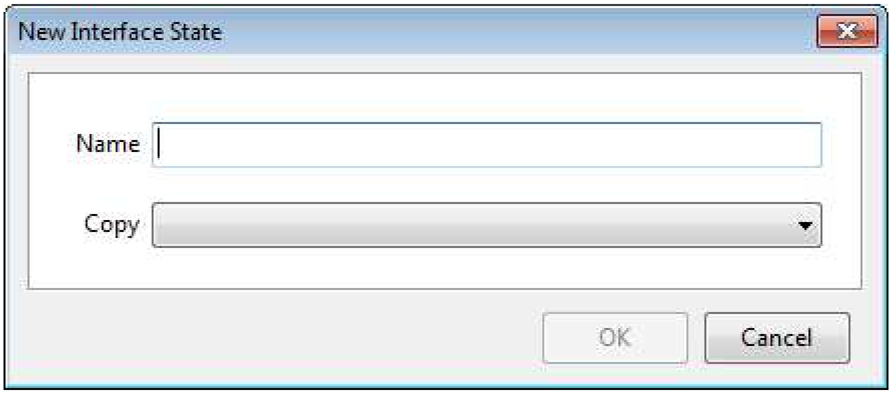
Adding a New Interface State

Control interfaces consist of one or more interface states. The default state is, by convention, named "Normal" and defined as the first interface state in the list of interface states.

Use the following steps to add a new interface state to the control interface:

- Click on the Add State toolbar button:

- The New Interface State dialog will now appear:



- In this dialog, specify the name of the interface state that you wish to create in the first edit control.

  Optionally, next, select an existing interface state to copy by using the combo box provided.

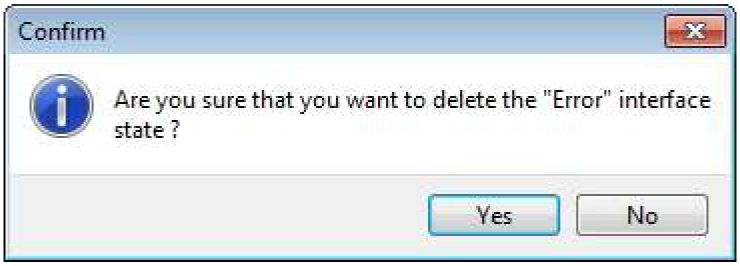
- Click on the OK button. A new interface state with the specified name will now appear in the list of defined interface states, and this interface state will be the selected interface state. If you copied an existing interface state, then the copied interface elements will appear in the middle element designer. If you did not copy an existing interface state, then a default "Base" element will appear in the element designer.

Removing an Existing Interface State

Use the following steps to remove an existing interface state from the control interface:

- Click on the Remove State toolbar button:

- A dialog similar to the following will now appear:



Confirm

Are you sure that you want to delete the "Error" interface state ?

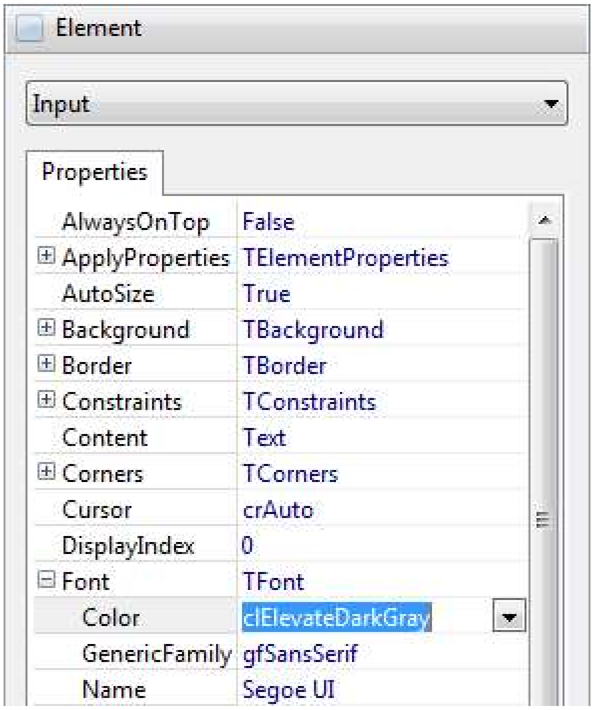
The name of the specifed interface state will reflect the interface state being removed. Click Yes to remove the selected interface state, or No to cancel the removal of the interface state.

Moving an Interface State

You can use drag and drop operations with the mouse to move an interface state to a different position in the list of defined interface states. Simply click on the desired interface state with the left mouse button, hold the left mouse button down, and drag the interface state to the desired new position.

Element Inspector

The element inspector is located on the right-hand side of the control interface editor, and allows you to modify the properties of the currently-selected element in the element designer. It consists of an element selection combo box and a list of the properties of an element:



AlwaysOnTop	False
⊞ ApplyProperties	TElementProperties
AutoSize	True
⊞ Background	TBackground
⊞ Border	TBorder
⊞ Constraints	TConstraints
Content	Text
⊞ Corners	TCorners
Cursor	crAuto
DisplayIndex	0
⊟ Font	TFont
Color	clElevateDarkGray
GenericFamily	gfSansSerif
Name	Segoe UI

To modify any property of an element, click on the desired property value, and type in the new value. If applicable, the property may have a special property editor in the form of a drop-down list or dialog that is accessible using a button to the right of the property value. Double-clicking on the property value will also automatically launch the applicable property editor.

You can get context-sensitive help on any property in the element inspector by clicking on the desired property and hitting the F1 key. For more information on using the help browser, please see the Accessing Help topic.

Adding a New Element to an Interface State

Control interface states consist of one or more interface elements. The default element is, by convention, named "Base" and defined as the base container element for the interface state.

Use the following steps to add a new element to an interface state:

- Click on the Add Element toolbar button:

- Click on the client area of an element in the element designer. The client area of an element is the area inside of the borders for the element.

Selecting an Element

To select a single element in the element designer, click on the desired element with the left mouse button. To select more than one element, hold down the Shift key while clicking on the desired elements with the left mouse button. Selecting multiple elements is desirable when one wants to resize or align multiple elements at the same time to ensure that their placement or size is uniform, or when one wants to copy and paste a group of elements.

> **Note**
> Any time you hover the mouse over any element, tooltip information will be displayed about the element, including the name and position/size.

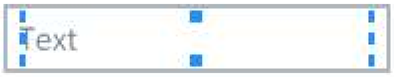
You can also use the mouse to directly select a group of elements using a lasso:

- If the group of elements are placed on the base element itself, then you can click and hold down the left mouse button to begin the selection. Then, while keeping the left mouse button down, move the mouse to lasso the desired element(s).

- If the group of elements are placed on a child element, then you can click and hold down the left mouse button, while also pressing the Ctrl key, to begin the selection. Then, while keeping the left mouse button and Ctrl key down, move the mouse to lasso the desired element(s).

Resizing an Element

Once an element has been placed on the active element's client area, you will see that the element will have designer handles on all four sides and corners of the element:



These designer handles can be used to change the origin and size of an element. To accomplish this, click on a designer handle with the left mouse button, hold the left mouse button down, and drag the designer handle in the desired direction. You can also use the keyboard to resize an element by holding down the Shift key while using the up, down, right, and left arrow keys to resize the element on a pixel-by-pixel basis.

Note

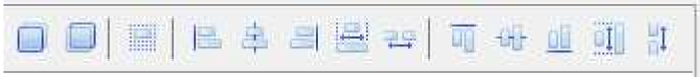
Certain elements may have constraints on how tall/wide they can be. In such cases, attempts to resize the element will result in the element size not exceeding the specified constraints. Also, you cannot use the left mouse button to resize elements when multiple elements are selected. In such cases, you can only use the keyboard to do so (Shift+Arrow Keys).

Moving an Element

To move an element, click on the element with the left mouse button, hold the mouse button down, and drag the element to the desired location. You can also use the keyboard to move an element by holding down the Ctrl key while using the up, down, right, and left arrow keys to move the element on a pixel-by-pixel basis. Both of these techniques also work when multiple elements are selected.

Element Layout and Alignment

The layout toolbar on the element designer can be used to adjust the alignment and layering (send to back/bring to front) of elements:



Each layout toolbar button has tooltip help that explains the purpose of the button.

Deleting an Element

To delete an element, select the desired element in the element designer and hit the Delete key or click on the Remove Element toolbar button:

This will also work when multiple elements are selected.

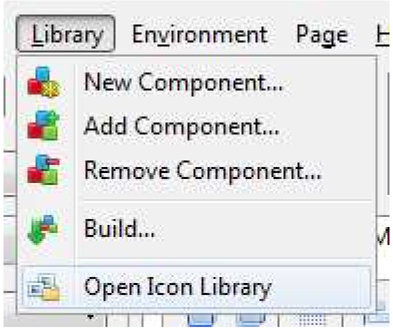
Warning

Undo functionality is currently not available for the element designer, so any modifications or deletions of elements cannot be undone. Please be careful when deleting elements to ensure that one does not lose a lot of hard work. If you do accidentally delete an element, you can fix the issue by simply closing the interface without saving the modifications, and then re-opening the interface. However, this depends upon how much other work has been done to the interface since the last save point, so it is wise to save your modifications on a regular basis.

2.25 Opening the Icon Library

Use the following steps to open the icon library:

- Click on the **Library** option in the main menu. The Library menu will open.
- Click on the **Open Icon Library** option in the Library menu:



- The icon library will now be opened in the Control Interface Editor.

Note
The icon library that is opened is dependent upon the active project and whether there exists a customized version of the icon library in one of the active project's search paths. If there are no customized icon libraries in the search paths for the active project, then the default icon library that ships with Elevate Web Builder will be opened.

If you want to customize the icon library that ships with Elevate Web Builder for the active project, simply use the **File/Save As** menu option to save the default icon library interface file in a different folder/directory. You should always use the default interface file name "TIconLibrary.wbi", even for customized icon libraries. If you do not use the correct interface file name, then the customized icon library will be ignored.

This page intentionally left blank

Chapter 3
Using Visual Controls

3.1 Standard Controls

The following are the visual controls in the standard component library included with Elevate Web Builder. They are grouped and ordered by the category in which they are installed and displayed on the component palette in the IDE.

Standard

The standard controls are those commonly used for the display and editing of data, and most of them are capable of being bound to a dataset. Please see the Binding Controls to DataSets topic for more information.

Control	Description
TLabel	Label control
THTMLLabel	HTML label control
TBalloonLabel	Balloon label control
TAlertLabel	Alert label control
TButton	Button control
TDialogButton	Dialog button control
TIconButton	Icon button control
TCheckBox	Check box control
TRadioButton	Radio button control
TEdit	Single-line edit control
TPasswordEdit	Single-line password edit control
TMultiLineEdit	Multi-line edit control
TListBox	List box control
TCalendar	Calendar control
TButtonComboBox	Button combo box control
TEditComboBox	Editable combo box control
TDateEditComboBox	Editable date combo box control
TDialogEditComboBox	Editable dialog combo box control
TFileComboBox	File upload combo box control
TGrid	Grid control

Graphics

Graphic controls are used for displaying images or providing a canvas for drawing/painting operations:

Control	Description
TImage	Image control
TIcon	Icon control
TAnimatedIcon	Animated icon control
TPaint	Painting control with canvas
TSlideShow	Slide-show control

Indicators

Indicator controls show progress and other types of graphic information:

Control	Description
TProgressBar	Progress bar control

Multimedia

Multimedia controls allow the playback of both audio and video:

Control	Description
TAudio	Audio playback control
TVideo	Video playback control

Containers

Container controls are used to group together controls within a parent control:

Control	Description
THeaderPanel	Header panel control
TScrollPanel	Scrollable panel control
TBasicPanel	Basic panel control
TGroupPanel	Group panel control with caption
TPanel	Panel control with caption bar
TPagePanel	Paged panel control
TSizer	Sizer control

Menus

Menu controls are used for displaying a list of focusable and selectable menu items:

Control	Description
TMenu	Menu control
TMenuBar	Menu bar control

ToolBars

Toolbar controls are groups of non-focusable buttons contained within a parent control:

Control	Description
TToolBar	Toolbar control
TDataSetToolBar	Dataset toolbar control

Browser

Browser controls are encapsulations of various types of legacy browser functionality:

Control	Description
THTMLForm	HTML form control
TLink	Link control
TBrowser	HTML document display control
TPlugin	Plugin control
TMap	Google Maps control

3.2 Creating and Showing Forms

Before using any form classes, you must first create an instance of the form class, which you can do at design-time or at run-time:

Creating a Form at Design-Time

The easiest way to create a form is by using the IDE to create a new form. When a form is created at design-time in the IDE, it is automatically designated as an auto-create form in the application project, which means that the form will automatically be created during application startup. The first form in the list of auto-create forms is considered the main form of the application, and will also be shown at application startup. For example, the following shows the main program source of an application that has several auto-create forms:

```
project FormSubmit;

contains Main, Results;

uses WebForms, WebCtrls;

begin
   Application.Title := 'HTML Form Submittal Example';
   Application.CreateForm(TMainForm);
   Application.CreateForm(TResultsDialog);
   Application.Run;
end.
```

Because the TMainForm form class is the first form class in the list of auto-create forms, it is considered the main form of the application and will automatically be shown when the Application Run method is executed.

Please see the Adding to an Existing Project topic for more information on adding a new form to a project.

Creating a Form at Run-Time

You can also create a form instance at run-time using code. This is useful for forms that are not used very often and for which having them auto-created would be a waste of memory. The following is an example of creating a form and showing it (modally) at run-time:

```
uses ProgFrm;

procedure TMainForm.LaunchButtonClick(Sender: TObject);
begin
   ProgressForm:=TProgressForm.Create(nil);
   ProgressForm.ShowModal;
end;
```

Note
In the above example the ProgressForm variable is declared in the interface section of the TProgressForm's unit (ProgFrm).

Showing and Hiding a Form

The TForm Show and ShowModal methods will show a form in a non-modal and modal fashion, respectively. See below for more information on modal forms.

Showing a form will also cause the form to be brought to the front of the visual stacking order via the BringToFront method.

To hide a form, call the TForm Hide method, which simply toggles the visibility of the form. In order to close the form and trigger the TForm OnCloseQuery and OnClose events, call the Close method instead.

Hiding or closing a form will also cause the form to be sent to the back of the visual stacking order via the SendToBack method.

Modal Forms

When a form is shown modally, the application displays a modal overlay over the entire surface and all other forms that prevents any keyboard or mouse input for any form other than the current modal form.

You can use the Application.Surface.ModalOverlay.CloseOnClick property to enable/disable the ability to close all visible modal forms by simply clicking on the modal overlay.

Modal forms behave very differently in a web browser environment than in a Windows desktop environment, requiring modal dialogs/forms be coded differently. To use the above example again, this is what the example would look like in a traditional Windows desktop application when using a product like Delphi:

```
uses ProgFrm;

procedure TMainForm.LaunchButtonClick(Sender: TObject);
begin
   ProgressForm:=TProgressForm.Create(nil);
   try
      ProgressForm.ShowModal;
   finally
      ProgressForm.Free;
   end;
end;
```

If you were to run the above code in a web browser, you would probably see a slight flicker as the form was shown and then immediately freed. This is because the ShowModal method, or any method in the JavaScript execution environment, does not cause the code execution to yield. Thus, the ShowModal method is called, and then the Free method is called right after the form is shown.

Because of the lack of the ability to yield execution, such forms must be closed/freed using a technique involving creating an event handler for the TForm OnClose event from the **calling** form and assigning that event handler to the OnClose event of the form that needs to be responded to. The following example shows how this would be done:

```
uses ProgFrm;

procedure TMainForm.ProgressFormClose(Sender: TObject);
begin
   ProgressForm.Free;
end;

procedure TMainForm.LaunchButtonClick(Sender: TObject);
begin
   ProgressForm:=TProgressForm.Create(nil);
   ProgressForm.OnClose:=ProgressFormClose;
   ProgressForm.ShowModal;
end;
```

This is quite a departure from the way that the OnClose event handler is used in desktop applications. With desktop applications, the form's OnClose event is normally assigned an event handler that is defined within the form being closed. If one simply remembers that the TForm OnClose event is a "special" event in this regard, then the concept will be easier to remember and implement properly in one's applications.

Form Events

The TForm OnCreate event is fired while a form is being created and is an ideal place to perform any initialization processing for the form.

The TForm OnCloseQuery event is fired when an attempt to close (hide) the form occurs. To prevent the close from occurring, return False as the result in an event handler for this event.

The TForm OnClose event is fired after the form is closed (hidden).

The TForm OnDestroy event is fired before a form is destroyed, and is an ideal place to dispose of any resources that need to be disposed of before the form is destroyed.

3.3 Showing Message Dialogs

Message dialogs are critical in a visual application for displaying important messages such as errors or warnings to users, as well as asking the user to answer important questions that determine the overall flow of processing. There are two procedures that provide the message dialog functionality in Elevate Web Builder:

- ShowMessage - This procedure simply displays a message to the user using a modal dialog containing the message and a single OK button.

- MessageDlg - This procedure displays a message to the user using a modal dialog containing the message and any number of user-configured buttons.

The ShowMessage procedure is the simplest to use when you only want to display a message to the user and do not need to ask the user to provide any further information. The following example shows how you would show such a message dialog:

```
function CheckTrial: Boolean;
begin
   if IsTrialVersion and (TrialDaysLeft > 0) then
      begin
      ShowMessage('You are using a trial version and have '+
                  IntToStr(TrialDaysLeft)' evaluation days '+
                  'left until your trial version expires.');
      Result:=True;
      end
   else
      begin
      ShowMessage('Your trial version has unfortunately expired.');
      Result:=False;
      end;
end;
```

The MessageDlg procedure is more versatile, but also slightly more complicated. It allows you to specify various attributes of the modal dialog used to display the message such as the dialog caption, the type of dialog (determines the icon used for the dialog), which buttons to display on the dialog, and whether or not to display a dialog close button. The following example shows how you would use this procedure to ask the user to confirm the deletion of a customer order in a dataset:

```
procedure TMasterDetailForm.CheckDelete(DlgResult: TModalResult);
begin
   if (DlgResult=mrYes) then
      begin
      Database.StartTransaction;
      CustomerOrders.Delete;
      Database.Commit;
      end;
end;

procedure TMasterDetailForm.DeleteOrderButtonClick(Sender: TObject);
begin
   MessageDlg('Are you sure that you want to delete the '+
              CustomerOrders.Columns['OrderID'].AsString+' order placed on '+
              CustomerOrders.Columns['OrderDate'].AsString+' ?','Please
      Confirm',
              mtConfirmation,[mbYes,mbNo],mbNo,CheckDelete,True);
end;
```

Note
The MessageDlg procedure is overloaded and has two different versions. The first does not include the default button parameter after the array of button types, whereas the second version (shown above) does include the default button parameter.

As discussed in the previous Creating and Showing Forms topic, modal forms require some special event handler logic in order to execute code when the modal form is closed. This is especially true with message dialogs, which are always shown modally, and is why the MessageDlg procedure accepts a method reference as a parameter. The method reference should point to a method that matches the following type:

```
TMsgDlgResultEvent = procedure (DlgResult: TModalResult) of object;
```

When the modal message dialog form is closed, the event handler will be called and the type of button that the user clicked in the message dialog will be passed as the first parameter.

Note
The TModalResult message dialog result type is different from the button types (TMsgDlgBtn type) that are passed as an array parameter to the MessageDlg procedure. The two types are similar, but there are additional results such as mrNone, which indicates that the user closed the dialog without clicking on any button at all.

3.4 Showing Progress Dialogs

Progress dialogs are critical in a visual application for displaying progress while the application is executing code or the application is waiting on an event handler to complete, such as an event handler for the TServerRequest OnComplete event. There are two procedures that provide the progress dialog functionality in Elevate Web Builder:

- ShowProgress - This procedure simply displays an animated icon and a message to the user using a modal dialog containing the message.

- HideProgress - This procedure hides any active progress dialog.

Warning
The ShowProgress and HideProgress procedures are reference-counted, so you should always ensure that you call the HideProgress procedure as many times as you call the ShowProgress procedure.

The following example shows how you would show such a progress dialog:

```
procedure TMainForm.LoadCustomers;
begin
   ShowProgress('Loading customers...');
   Customer.AfterLoad:=CustomerAfterLoad;
   Customer.OnLoadError:=CustomerLoadError;
   Database.LoadRows(Customer);
end;

procedure TMainForm.CustomerAfterLoad(Sender: TObject);
begin
   HideProgress;
end;

procedure TMainForm.CustomerLoadError(Sender: TObject; const ErrorMsg:
      String);
begin
   HideProgress;
   MessageDlg(ErrorMsg,'Error Loading Customers',mtError,[mbOk]);
end;
```

3.5 Using HTML Forms

HTML forms in Elevate Web Builder are represented by the THTMLForm component. HTML forms are the legacy way of allowing a user to input information into various controls on a form and have that information sent to the web server using an HTTP POST request. The THTMLForm component is a simple container control, which gives you the option of having multiple sub-forms within the same visual form, each with its own ability to submit information independently of the other.

Input Controls

The following standard controls can be used to input information that can be sent as part of the form submittal process:

Control	Description
TEdit	Single-line edit control
TPasswordEdit	Single-line password edit control
TMultiLineEdit	Multi-line edit control
TCheckBox	Check box control
TRadioButton	Radio button control
TListBox	List box control
TCalendar	Calendar control
TButtonComboBox	Button combo box control
TEditComboBox	Editable combo box control
TDateEditComboBox	Editable date combo box control
TDialogEditComboBox	Editable dialog combo box control
TFileComboBox	File upload combo box control

Note
These are only the standard controls included with Elevate Web Builder's standard component library, so this list does not include any installed 3rd party controls that may also allow usage with an HTML form.

Submitting the Input Information

In order to actually submit the input information as an HTTP POST request to the web server, complete the following steps:

- Make sure that the THTMLForm's Encoding property is set to feMultiPartFormData. You can use other encoding types, but this is the default and supports the most common type of form submission, including submitting files using the TFileComboBox control.

- Make sure that the THTMLForm's Method property is set to fmPost. This is the default value, so you'll probably never need to change this property.

- Make sure that the THTMLForm's URL property is set to the desired URL.

- Call the THTMLForm's Submit method to perform the submission. When the HTML form is submitted, all input controls contained within the HTML form are included, and the names used for the HTML form variables that are submitted are the same as the Name property of the included controls.

Testing Form Submittals

The internal web server embedded in the IDE includes support for echoing back any HTML form variables submitted using the Submit method. Just be sure to use the following URL for the THTMLForm's URL property:

```
http://localhost/formsubmit
```

Note
The above URL assumes that the internal web server is listening on the standard port 80. Please see the Modifying Environment Options topic for more information on configuring the internal web server.

Redirecting Form Submittal Output

By default, the THTMLForm Submit method will direct any response from the web server to a special hidden frame that Elevate Web Builder includes to suppress any output from the submittal. This is done to prevent the web browser from navigating away from the Elevate Web Builder application itself. If you want to display the output from the HTML form submittal process, or track when the submittal is completed, you can use the THTMLForm's Output property to do so. This property allows you to specify a TBrowser control that will receive the web server response to the HTML form submittal. In addition, you can assign an event handler to the TBrowser OnLoad event to determine when the web server response has been loaded into the frame encapsulated by the TBrowser control.

3.6 Layout Management

The layout management functionality in Elevate Web Builder handles all aspects of the layout of controls at design-time and run-time. Layout management is available for all controls in the component library, including the application surface and forms.

Control Layout Properties

Each control in an Elevate Web Builder application possesses several key properties that control the layout of the control:

Left, Top, Width, and Height

The TControl Left, Top, Width, and Height properties specify the **defined** position and dimensions of the control. These property values serve as the basis for the layout of the control, but can be modified by other layout properties such as the Layout and Constraints properties (see below).

Layout Order

The TControl LayoutOrder property of a control specifies the integer position of the control relative to any and all other child controls within the same container control. The layout order, as its name implies, determines how controls are positioned, relative to one another, by the layout functionality.

Layout

The TControl Layout property of a control is a class instance property that specifies several key aspects of the layout for the control via the following properties:

Layout Property	Purpose
Position	Specifies the type of positioning, if any, to use for the control within the layout rectangle of its container control.
Stretch	Specifies a stretch direction, if any, to apply to the control.
Consumption	Specifies the direction in which the control consumes space and modifies the layout rectangle for its container control, if at all.
Reset	Allows a control to reset the layout rectangle for its container.
Overflow	Allows a control to specify the direction in which a layout rectangle should automatically be adjusted when the control's dimensions exceed one of the sides of the layout rectangle.

Note
 Please see the section entitled Layout Rectangle below for more information on the concept of the layout rectangle.

Constraints

The TControl Constraints property of a control specifies any minimum and maximum constraints on the width and height of the control.

Margins

The TControl Margins property of a control specifies any margins for the control.

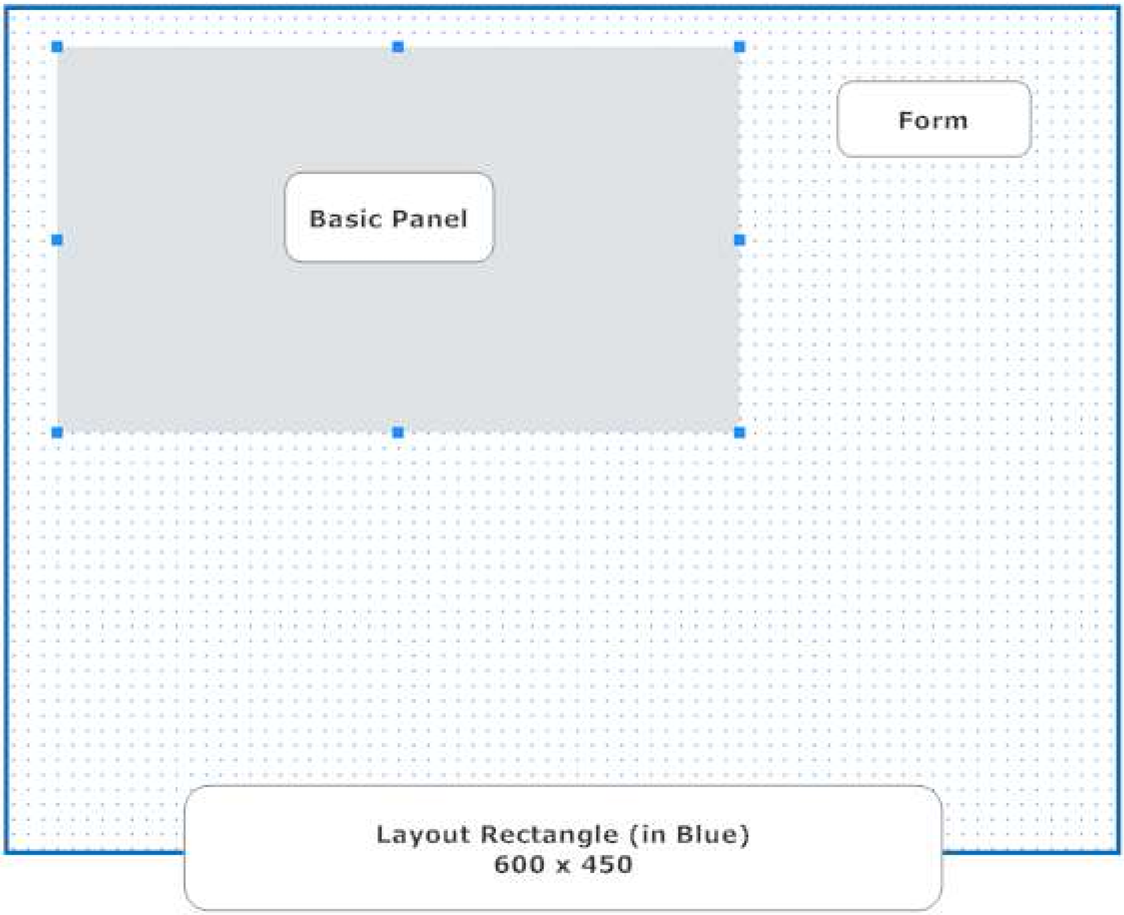
Padding and Borders

The padding and borders of a control vary depending upon the control class. Some control classes expose one or both of these properties, while others do not. However, these properties **do** affect the layout of any child controls contained within a container control by reducing the size of the layout rectangle for the container control.

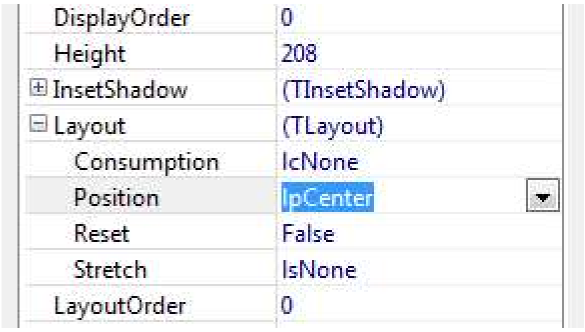
Layout Rectangle

In order to understand how the layout management works in Elevate Web Builder, it is important to understand the concept of the layout rectangle. The layout rectangle represents the area of a container control in which the layout of a child control is taking place. The layout rectangle is not a static area: each child control may consume space in the layout rectangle in a specific direction, thus reducing its size, and the layout rectangle can be segmented into different areas via reset points. The layout rectangle is initialized to the client rectangle for the container control. The client rectangle is defined as the bounding rectangle of a container control, minus the width of any borders or padding defined for the container control.

To illustrate the concept of the layout rectangle in its most basic form, let's place a single TBasicPanel control instance on a form (TForm-descendant instance). In this case, the form instance is the container control and the TBasicPanel instance is the child control. Because the form instance does not have any borders or padding defined, the client rectangle, and subsequently, the layout rectangle, is the same size as the form instance's bounding rectangle.

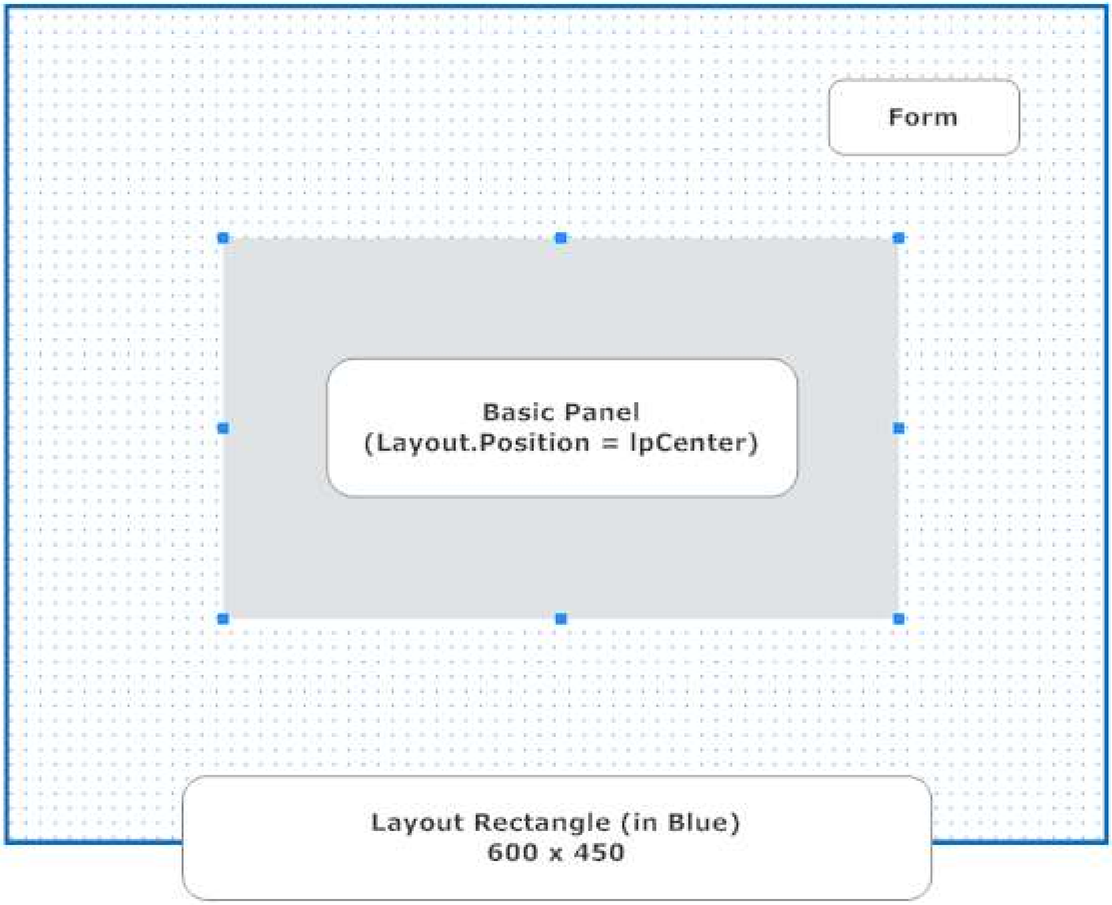


We'll specify that the Layout.Position property of the TBasicPanel should be **lpCenter**:



DisplayOrder	0
Height	208
⊞ InsetShadow	(TInsetShadow)
⊟ Layout	(TLayout)
Consumption	lcNone
Position	lpCenter
Reset	False
Stretch	lsNone
LayoutOrder	0

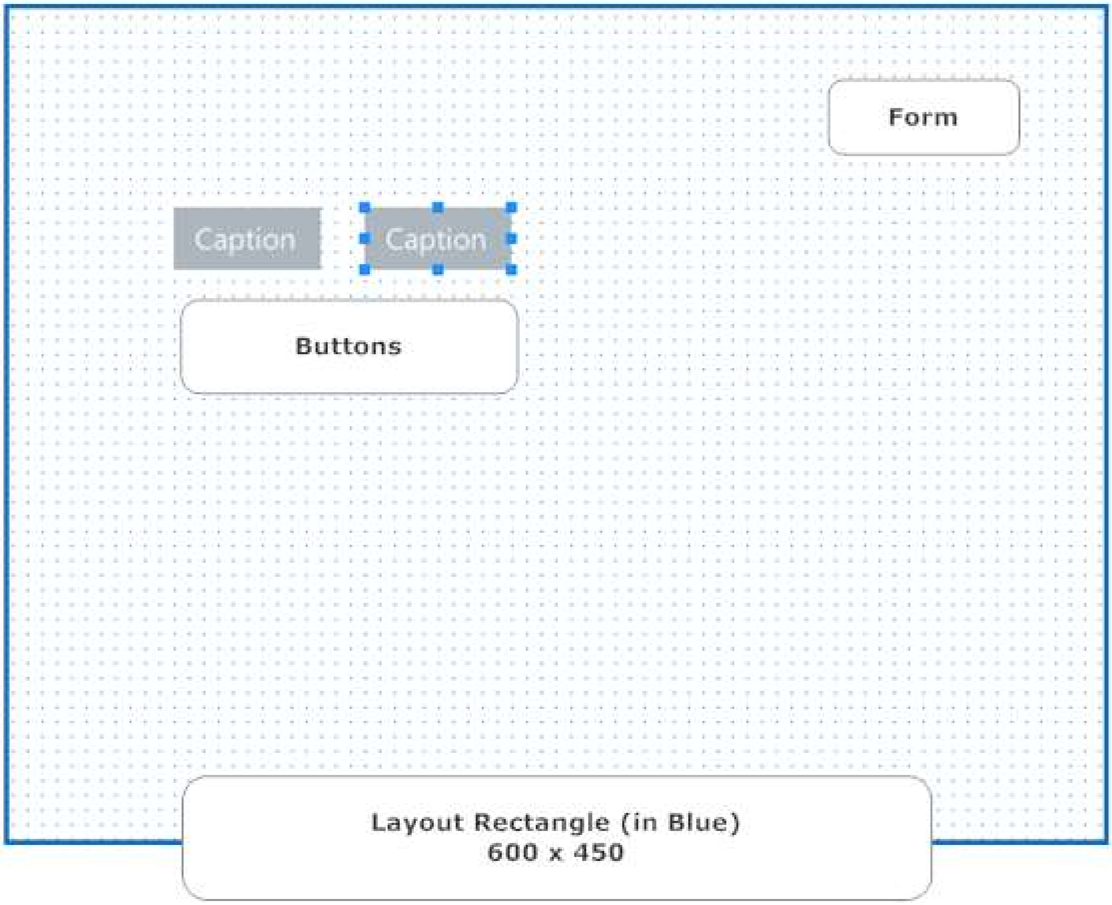
The resulting layout looks like this:



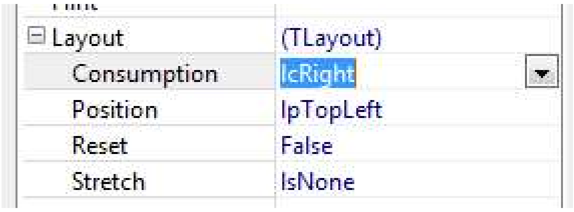
As you can see, the layout functionality used the layout rectangle of the form instance to center the defined dimensions of the TBasicPanel control instance. In this case, the layout rectangle was used for positioning only.

Consuming Space in the Layout Rectangle

To illustrate how space consumption affects the layout rectangle, let's place two TButton control instances on a form (default width of 80 pixels). Again, the form instance is the container control and the TButton instances are the child controls, and the initial layout rectangle is the same as the client rectangle of the form instance.

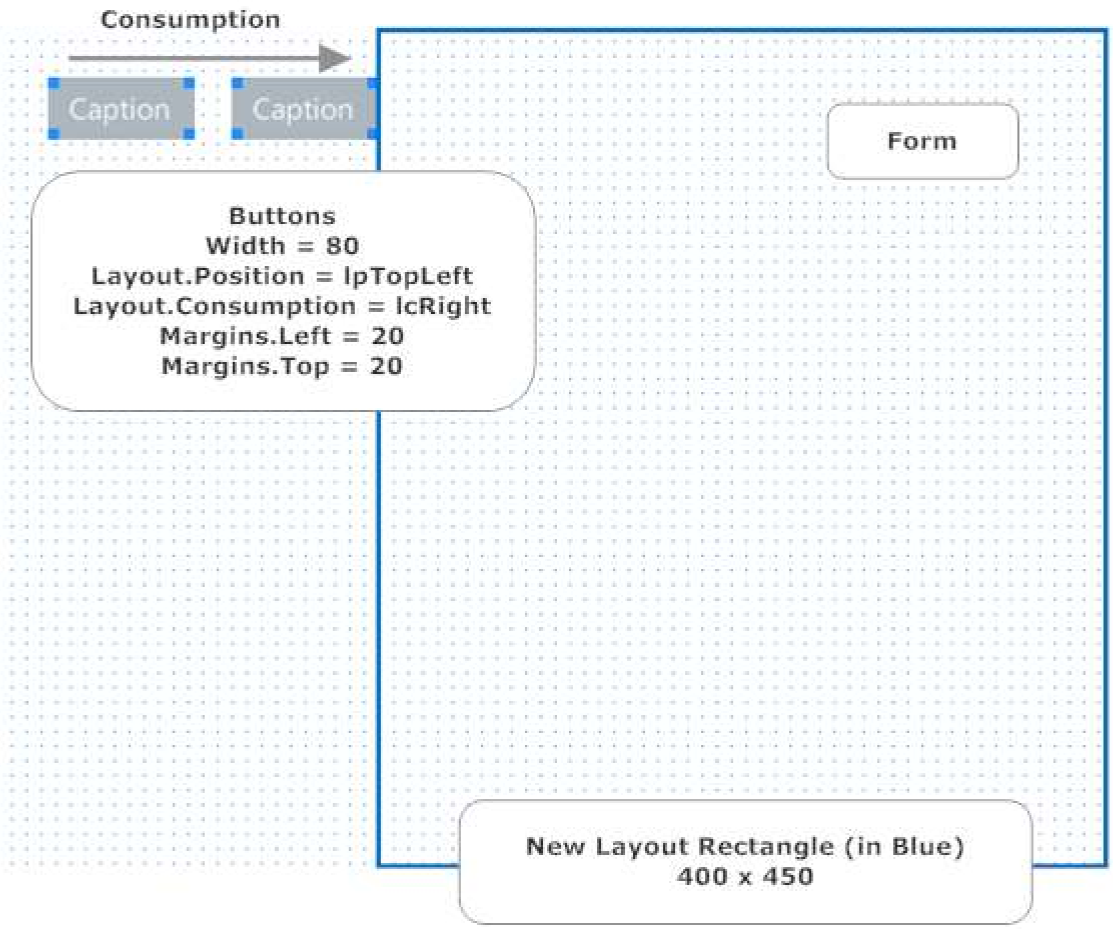


We'll specify that the Layout.Position property of both TButton instances should be **lpTopLeft**, the Layout.Consumption property should be **lcRight**, and the Margins.Left and Margins.Top properties should be set to **20 pixels** for proper spacing:



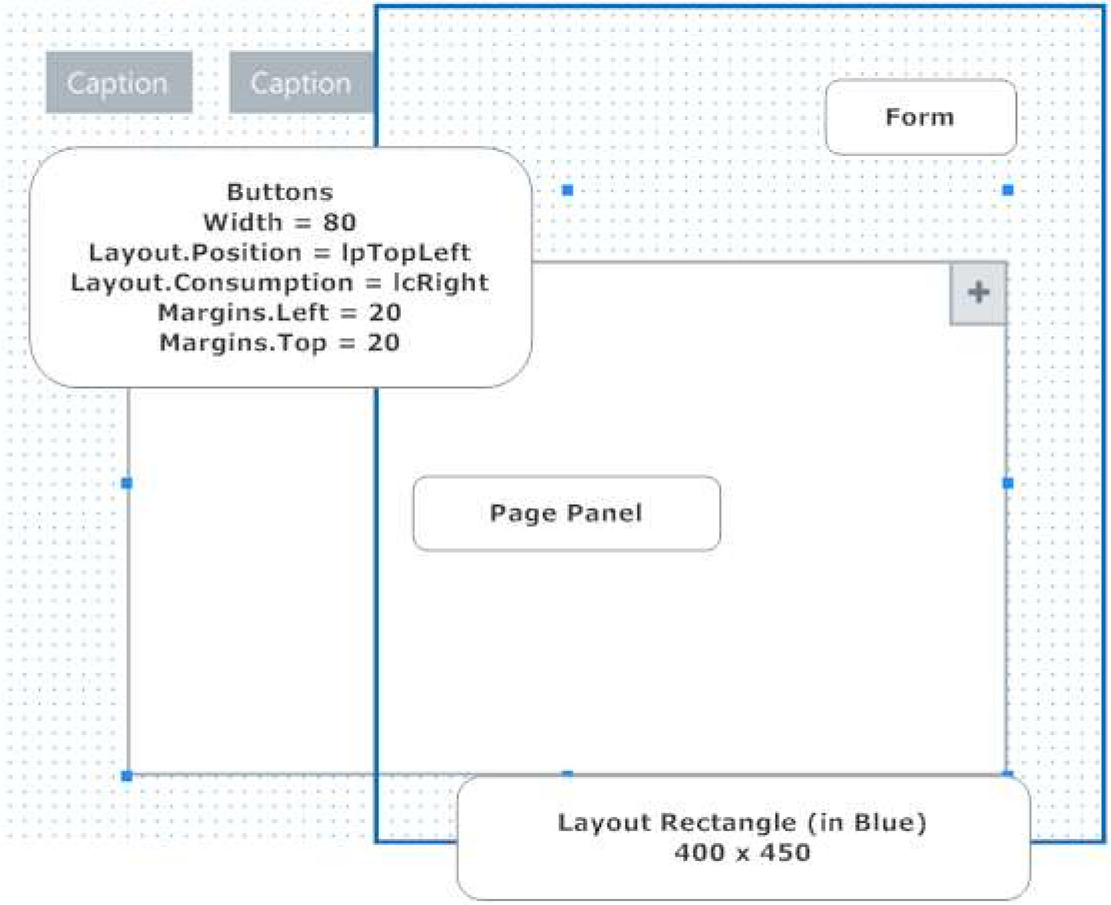
⊟ Layout	(TLayout)
Consumption	lcRight
Position	lpTopLeft
Reset	False
Stretch	lsNone

The resulting layout looks like this:

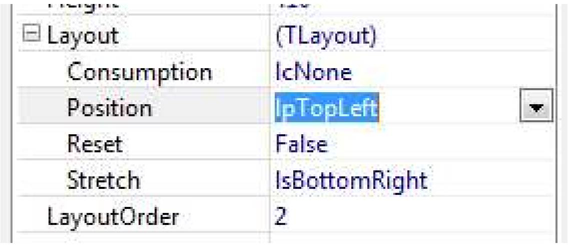


The layout functionality reduced the width of the layout rectangle by the width of each button (80 pixels) combined with the left margin of each button (20 pixels), for a total reduction of 200 pixels.

In most cases a form would not consist of just two buttons, so let's continue with the layout by placing a TPagePanel control instance on the form.

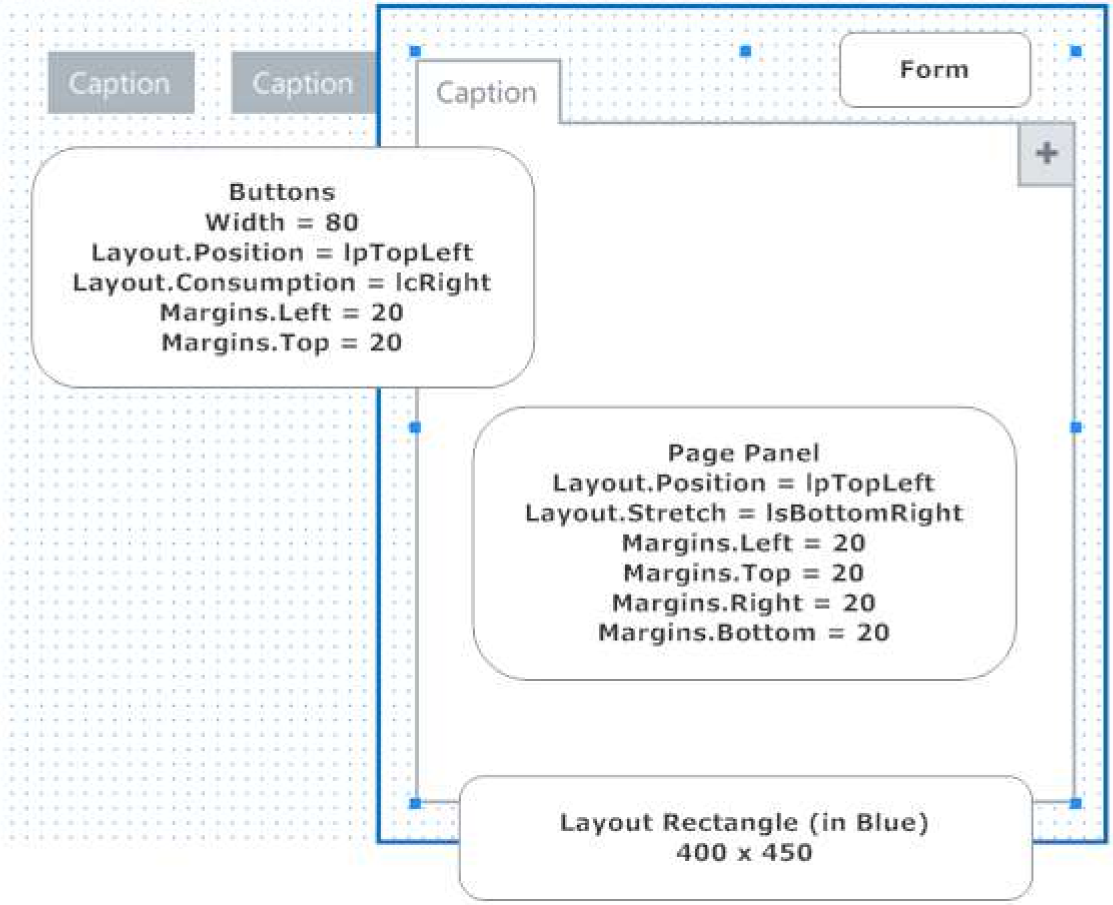


We want the TPagePanel instance to use the rest of the available space on the form **below** the two TButton instances, so let's specify that the Layout.Position property of the TPagePanel instance should be **lpTopLeft**, the Layout.Stretch property should be **lsBottomRight**, and the Margins.Left, Margins.Top, Margins.Right, and Margins.Bottom properties should be set to **20 pixels** for proper spacing:



⊟ Layout	(TLayout)
Consumption	lcNone
Position	lpTopLeft
Reset	False
Stretch	lsBottomRight
LayoutOrder	2

The resulting layout looks like this:



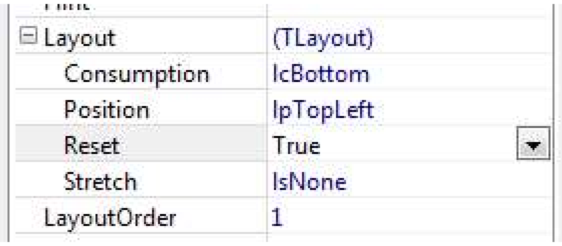
Buttons
Width = 80
Layout.Position = lpTopLeft
Layout.Consumption = lcRight
Margins.Left = 20
Margins.Top = 20

Page Panel
Layout.Position = lpTopLeft
Layout.Stretch = lsBottomRight
Margins.Left = 20
Margins.Top = 20
Margins.Right = 20
Margins.Bottom = 20

Layout Rectangle (in Blue)
400 x 450

As you can see, this is not exactly what we wanted, and the TPagePanel instance is to the right of the buttons instead of below the buttons.

To fix this, we only need to change two properties for the second TButton instance: we need to specify that the Layout.Consumption property should be **lcBottom** and that the Layout.Reset button should be **True**:



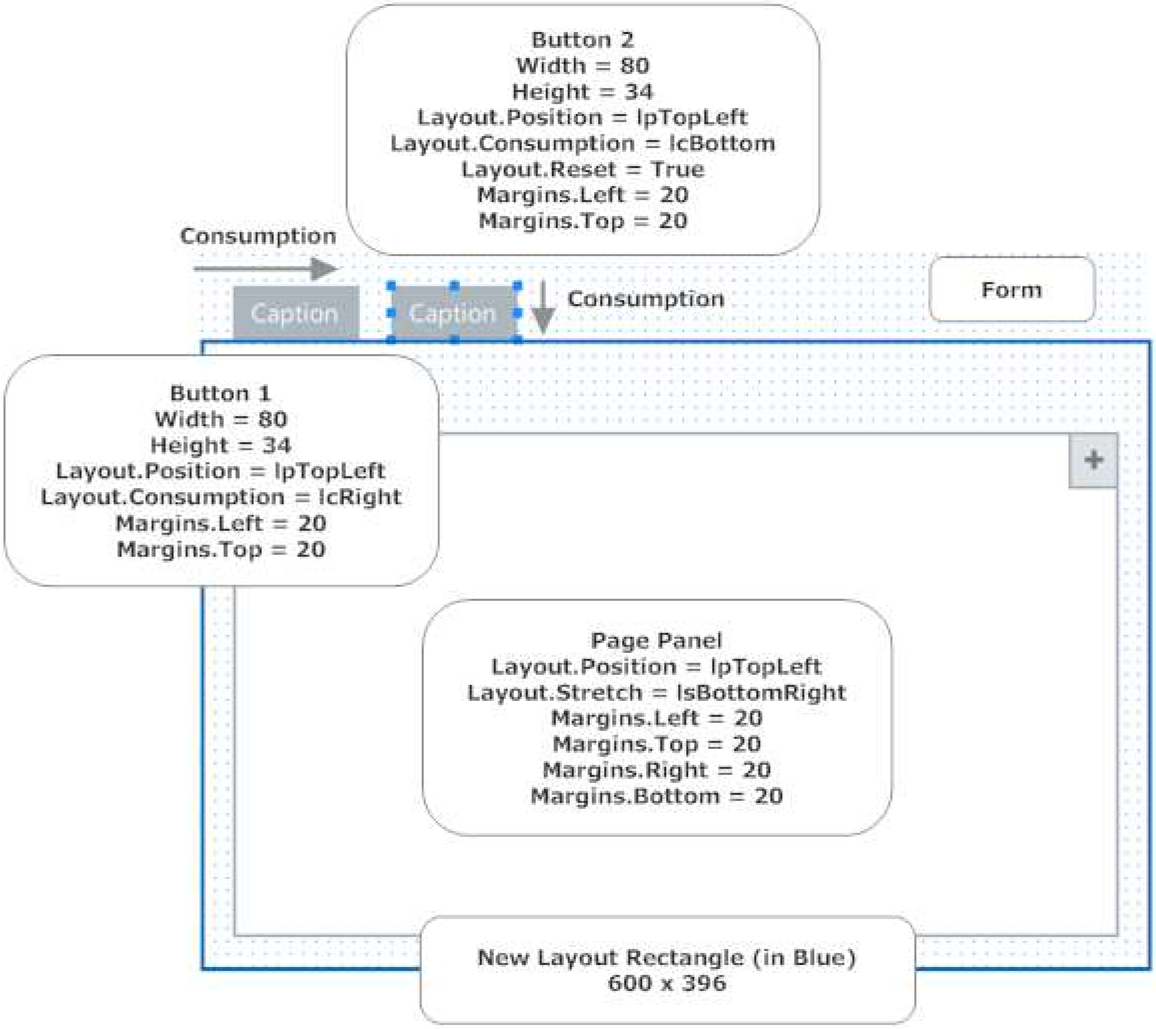
⊟ Layout	(TLayout)
Consumption	lcBottom
Position	lpTopLeft
Reset	True
Stretch	lsNone
LayoutOrder	1

Changing these two properties in this manner does two things:

- It changes the consumption direction towards the bottom of the layout rectangle, which is where we want the TPagePanel instance to be placed.

- It resets the layout rectangle back to the last reset point. Since this is the only control whose Layout.Reset property is set to True, this means that the last reset point is the original layout rectangle for the form. The reset of the layout rectangle will occur **before** the control consumes any space.

The resulting layout looks like this:



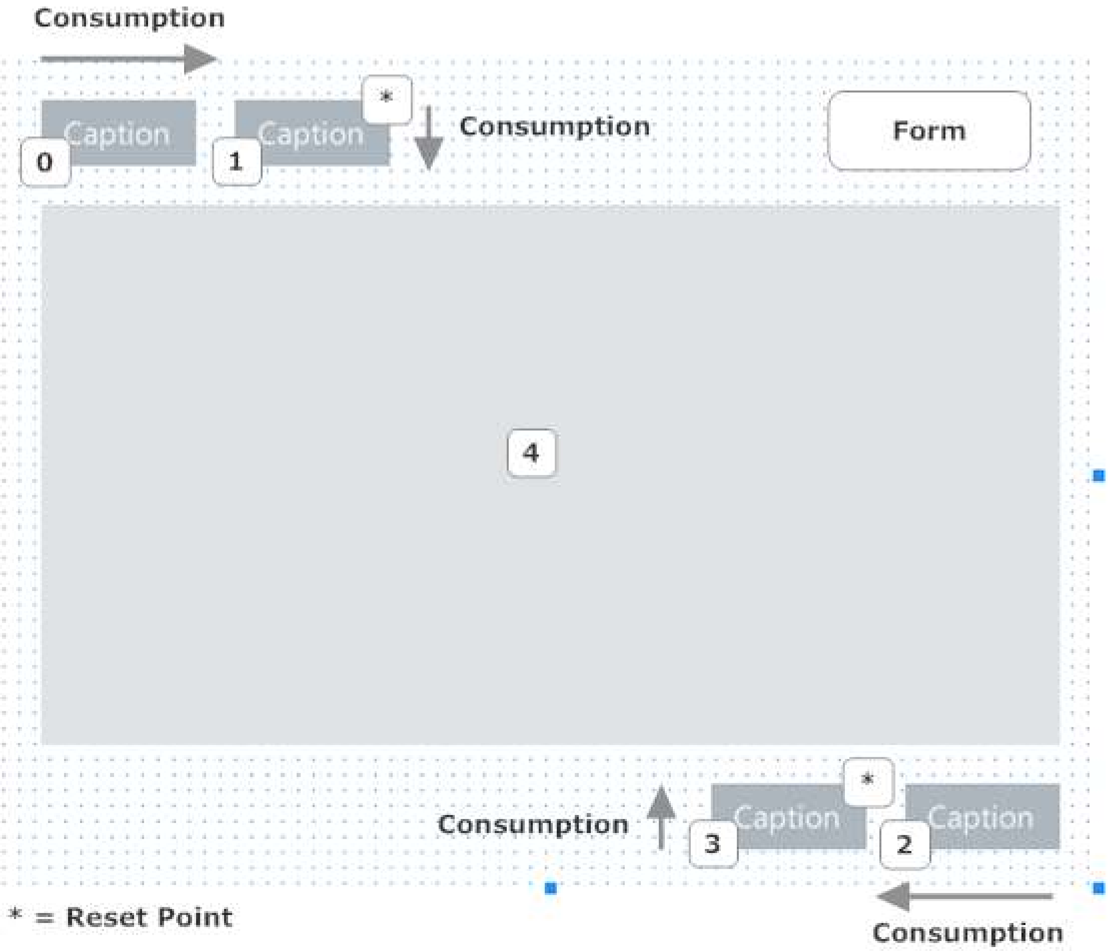
With these changes, the layout functionality reduced the height of the layout rectangle by the height of the second button (34 pixels) combined with the top margin of the second button (20 pixels), for a total reduction of 54 pixels.

> **Note**
> You'll also notice that we did not specify the Layout.Consumption property for the TPagePanel instance. This is because consumption only affects the positioning of controls that come **after** the current control in the layout order. Since the TPagePanel instance is the last control placed on the form, there is no point in specifying the Layout.Consumption property.

Reset Points

Reset points are useful for situations where you have a series of controls consuming space in one direction according to their layout order, but wish to change the consumption direction after the last control in the series. Reset points are set by setting a control's Layout.Reset property to True. As mentioned above, when a reset point is encountered the layout rectangle is set to the layout rectangle of the **last** reset point. This reset point layout rectangle represents the layout rectangle **after** any space consumption took place for the control setting the reset point. If there were no prior reset points, then the layout rectangle is set to the client rectangle of the container control.

The following layout shows how you can use multiple reset points to arrange several series of controls without needing to use special container controls:



The numbers represent the LayoutOrder property value for the control, and the asterisks (*) represent where a control has its Layout.Reset property set to True. The controls at the top left all have their Layout.Position properties set to **lpTopLeft**, and the controls at the bottom right all have their Layout.Position properties set to **lpBottomRight**. The control in the middle has its Layout.Position property set to **lpTopLeft**, and its Layout.Stretch property to **lsBottomRight**.

Constraints and Stretching

The defined constraints for a control are **always** in effect, and any attempts to modify the dimensions of the control in a way that violates these constraints will result in the modification being adjusted so that it adheres to the applicable constraint. This makes constraints very useful when combined with the Layout.Stretch property options. For example, in many of the example applications included with Elevate Web Builder, you will see code like this:

```
procedure TMainForm.MainFormCreate(Sender: TObject);
begin
```

```
   Application.ViewPort.OverflowY:=otAuto;
   with Application.Surface do
      begin
      Constraints.Min.Height:=(Self.Height+40);
      Background.Fill.Color:=clElevateFillGray;
      end;
end;
```

Note
By default, the TSurface control interface is defined so that the application surface's Layout.Position property is lpTopLeft and the application surface's Layout.Stretch property is lsBottomRight.

This code specifies that any time the application surface vertically overflows the browser viewport, a vertical scrollbar should be shown in the browser. In addition, it sets the minimum size of the application's surface to 40 pixels taller than the main form. Combined with the fact that the surface is set to stretch to fill the entire browser viewport, these two settings enable the following behaviors:

- If the browser viewport is **larger** than the minimum surface height, the application surface will stretch to fill the browser viewport.

- If the browser viewport is **smaller** than the minimum surface height, the surface will remain the minimum height and the browser viewport will display a vertical scrollbar.

Layout Overflow and Responsive Layouts

The Layout.Overflow property of a control can be used to create responsive layouts by giving the developer the ability to specify how the current layout rectangle should be adjusted when the dimensions of the control exceed the left, top, right, or bottom bounds of the current layout rectangle. The layout management uses the Overflow property to determine which direction the **prior** (based upon the layout order) control's Consumption property should be **temporarily** adjusted in order to prevent the current control's dimensions from exceeding the bounds of the current layout rectangle. When an overflow condition occurs, the Reset property for the prior control is temporarily set to True, resetting the current layout rectangle to the layout rectangle of the last reset point, and the Consumption property for the prior control is temporarily modified according to the following rules:

Overflow	Consumption
loTop	lcTop
loLeft	lcLeft
loRight	lcRight
loBottom	lcBottom

This provides the developer the ability to specify an initial desired layout with positioning, stretching, consumption, margins, constraints, and reset points, but still allow the layout to adjust within a container control that may dynamically resize while the application is executing.

Note

It is important that you specify an Overflow property that makes sense for a given layout. For example, if the container control is a scrollable control, can be resized horizontally and vertically, but can only scroll vertically, then it would make no sense to specify an Overflow property value of loLeft or loRight. The same logic applies to a scrollable container control that can be resized horizontally and vertically, but can only scroll horizontally. With such a contaner control, it would make no sense to specify an Overflow property value of loTop or loBottom.

This page intentionally left blank

Chapter 4
Using Server Requests

4.1 Server Request Architecture

Elevate Web Builder produces web applications that are loaded once into a web browser. Such an application is different from a traditional web site with a collection of individual web pages that are navigated to using traditional URL links. In fact, navigating to a different URL in an Elevate Web Builder application will actually cause the application to be unloaded in the web browser, which is not the desired result for most situations.

Given this architecture, there needs to be a way for such an application to communicate with the web server in order to exchange data or content without causing an actual navigation or page load in the web browser. The name for this type of communication in modern web browsers is called AJAX, which stands for "Asynchronous JavaScript and XML". While AJAX was primarily designed to be used with XML data, it can be used with any type of textual content or data. Though AJAX can also be used in a synchronous, as opposed to asynchronous, manner, Elevate Web Builder always uses AJAX functionality in an asynchronous manner. What this means is that when a web server request is executed, the application will continue to execute and respond to user input while the request is being executed, and an event will be triggered when the request completes successfully or encounters an error.

When to Use Server Requests

It is important to understand when a server request should be used and, even more importantly, when one **shouldn't** be used. The following are cases where you should not use a server request:

- If you only need to load and display an image, use a TImage control instead.
- If you only need to send some values from input controls on a form to the web server, use the HTML Forms functionality instead.
- If you are using databases, then use the built-in database handling that is provided.

HTTP Server Requests

AJAX web server requests are basically equivalent to the requests that are made by a web browser on your behalf when navigating URL links in a web page. These requests use the HTTP protocol which determines how the request and its response from the web server are formatted. A typical HTTP request looks like this:

```
GET /testproject.html HTTP/1.1
Accept: text/html
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Cache-Control: max-age=0
Connection: keep-alive
Host: localhost
If-Modified-Since: Thu, 16 Aug 2012 18:35:21 GMT
```

```
User-Agent: Mozilla/5.0
```

Every HTTP request begins with a method name, followed by a URL and the version of the HTTP protocol being used by the web browser. Please see the following link for a complete definition of the various HTTP methods:

Method Definitions

Elevate Web Builder supports the GET, HEAD, POST, PUT, and DELETE methods in web server requests.

After the initial request line is one carriage return/line feed pair (0x0D and OxOA), followed by the request headers. All request headers use a format of:

```
<Header Name>: <Header Value>
```

Please see the following link for a complete definition of all standard HTTP headers:

Header Field Definitions

After the request headers are two carriage return/line feed pairs. If the request does not send any additional content, as would be the case with a POST request, then the request will not contain any additional data. If there is additional content, then the additional content will be sent after the two carriage return/line feed pairs. In addition, a "Content-Length" request header must be specified in the request headers that indicates the size, in characters, of the additional content.

> **Warning**
> If you do not specify a content length header, then the most likely result is that the web server will simply ignore the content, return an error code, or both.

For example, suppose that you want to send the following text content to a web server in a POST request:

```
The quick brown fox jumps over the lazy dog
```

The length of the text is 43 characters, so the POST request would look like this:

```
POST /postcontent HTTP/1.1
Accept: text/html
Accept-Encoding: gzip, deflate
Accept-Language: en-us
Cache-Control: max-age=0
Connection: keep-alive
Host: localhost
User-Agent: Mozilla/5.0
Content-Type: text/plain; charset=utf-8
Content-Length: 43
```

```
The quick brown fox jumps over the lazy dog
```

Note
At this point it is probably a good idea to point out that you do not have to format web server requests like this in order to use the server request functionality in Elevate Web Builder. However, it is important that you understand how such requests are formatted in order to properly add custom headers or content to web server requests, as well as to properly read and parse response content returned from the web server.

HTTP Server Responses

The format of responses from a web server are very similar to the format of the requests. A typical HTTP response from a web server looks like this:

```
HTTP/1.1 200 OK
Date: Thu, 17 Aug 2012 01:52:46 GMT
From: admin@elevatesoft.com
Server: Elevate Web Builder Web Server
Connection: Keep-Alive
Cache-Control: no-cache
Content-Type: text/plain; charset=utf-8
Content-Length: 139

NameEdit=Tim Young
EmailEdit=timyoung@elevatesoft.com
CommentsEdit=Comments
RememberMeCheckBox=False
```

Note
The response content is not necessarily representative of the content that may be returned by any web server request, and is only used to represent the response content as a simple key-value example.

Every HTTP response begins with the version of the HTTP protocol being used by the web server, followed by a numeric response code and a textual status message. Please see the following link for a complete definition of the various HTTP response codes:

Status Code and Reason Phrase

In both an Elevate Web Builder application, and an Elevate Web Builder web server module, there are defined constants that represent the common HTTP status codes. In an Elevate Web Builder application, you will find these constants in the WebHTTP unit, which contains the TServerRequest and TServerRequestQueue components (see below) and is part of the standard component library. In an Elevate Web Builder web server module, you will find these constants in the ewbhttpcommon unit, which is distributed as a .dcu (Delphi compiled unit) with the Elevate Web Builder Modules installlation.

The constants are defined as follows:

```
   HTTP_NONE = 0;
   HTTP_CONTINUE = 100;
   HTTP_OK = 200;
   HTTP_MOVED_PERMANENTLY = 301;
   HTTP_FOUND = 302;
   HTTP_SEE_OTHER = 303;
   HTTP_NOT_MODIFIED = 304;
   HTTP_MOVED_TEMPORARILY = 307;
   HTTP_BAD_REQUEST = 400;
   HTTP_NOT_FOUND = 404;
   HTTP_NO_LENGTH = 411;
   HTTP_INTERNAL_ERROR = 500;
   HTTP_NOT_IMPLEMENTED = 501;
   HTTP_SERVICE_UNAVAILABLE = 503;
```

Core Components

In Elevate Web Builder, the components that encapsulate the AJAX functionality in the web browser are:

TServerRequest
TServerRequest components can be dropped directly onto a visual form at design-time in a visual project, or created at run-time in both visual and non-visual projects. The TServerRequest component encapsulates a single web server request. The Method property specifies the HTTP method (default rmGet) and the URL property specifies the URL for the request. Although the web browser will automatically populate all required request headers, you can specify additional request headers using the RequestHeaders property. You can use the Execute method to actually execute the request.

TServerRequestQueue
TServerRequestQueue components can be dropped directly on a visual form at design-time in a visual project, or created at run-time in both visual and non-visual projects. The TServerRequestQueue component implements a queue of server requests in order to force serialization of the server requests so that requests are executed in the order in which they are added to the queue. For example, the TDatabase component uses an internal TServerRequestQueue component to ensure that dataset load requests, as well as transaction commit requests, are executed in the order that they are requested.

4.2 Executing a Server Request

The most common use for the TServerRequest component is to receive/send content to/from the web server. Elevate Web Builder does just that for loading the columns and rows for datasets, as well as committing database transactions and sending inserts, updates, and deletes to the web server. Datasets use the JSON format for exchanging data with the web server, but server requests do not impose any restriction on the format of the content that is sent or returned from the web server other than the fact that it must be textual (or encoded in a textual format, as is the case with Base64 encoding).

Use the following steps to execute a server request using a TServerRequest component:

- Make sure that the TServerRequest Method property is set to the desired value. The default value is rmGet.

- Assign the proper URL to the TServerRequest URL property.

> **Warning**
> If the origin (protocol, host, and port) specified in the URL is different than the origin for the application, then you will need to set the TServerRequest CrossOriginCredentials property to true in order to have any HTTP cookies and/or authentication headers sent to the web server that is servicing the HTTP requests for the URL.

- Assign any URL parameters to the TServerRequest Params property. The Params property is a TStringList object instance with an equals (=) name/value separator. Each parameter should be specified in the name=value format as a separate string in the list.

> **Note**
> URL parameters are automatically appended directly to the URL by the TServerRequest component when the Execute method is called, so do not add them directly to the URL property. You can use the RequestURL property to retrieve the full URL that will be sent to the destination web server when the server request is executed.

- Assign any custom request headers to the TServerRequest RequestHeaders property. The RequestHeaders property is a TStringList object instance with a colon (:) name/value separator. Each header should be specified in the following format as a separate string in the list:

```
Name: Value
```

- Create and assign an event handler to the TServerRequest OnComplete event. This will ensure that you can determine when the request is complete.

- Call the TServerRequest Execute method to initiate the web server request.

TServerRequest Example

For example, suppose that you wanted to retrieve customer data from the web server in the following key-value format:

```
ID=100
Name=ACME Manufacturing, Inc.
Contact=Bob Smith
Address1=100 Main Street
Address2=
City=Bedford Falls
State=NY
ZipPostal=11178
```

To do so, you would use code like the following:

```
procedure TMyForm.MyFormCreate(Sender: TObject);
begin
   MyRequest:=TServerRequest.Create(nil);
end;

procedure TMyForm.MyFormDestroy(Sender: TObject);
begin
   MyRequest.Free;
end;

procedure TMyForm.RequestComplete(Request: TServerRequest);
begin
   if (Request.StatusCode=HTTP_OK) then
      ShowMessage('The value of the customer ID is '+
                  Request.ResponseContent.Values['ID'])
   else
      raise EError.Create('Response Error: '+Request.StatusText);
end;

procedure TMyForm.GetCustomerClick(Sender: TObject);
begin
   MyRequest.URL:='/customer';
   MyRequest.Params.Add('method=info');
   MyRequest.ResponseContent.LineSeparator:=#10;
   MyRequest.OnComplete:=RequestComplete;
   MyRequest.Execute;
end;
```

TServerRequestQueue Example

To use the TServerRequestQueue component instead of the TServerRequest component, you would use the following code:

```
procedure TMyForm.MyFormCreate(Sender: TObject);
begin
   MyRequestQueue:=TServerRequestQueue.Create(nil);
end;

procedure TMyForm.MyFormDestroy(Sender: TObject);
```

```
begin
   MyRequestQueue.Free;
end;

procedure TMyForm.RequestComplete(Request: TServerRequest);
begin
   if (Request.StatusCode=HTTP_OK) then
      ShowMessage('The value of the customer ID is '+
                  Request.ResponseContent.Values['ID'])
   else
      raise EError.Create('Response Error: '+Request.StatusText);
end;

procedure TMyForm.GetCustomerClick(Sender: TObject);
var
   TempRequest: TServerRequest;
begin
   TempRequest:=MyRequestQueue.GetNewRequest;
   TempRequest.URL:='/customer';
   TempRequest.Params.Add('method=info');
   TempRequest.ResponseContent.LineSeparator:=#10;
   TempRequest.OnComplete:=RequestComplete;
   MyRequestQueue.AddRequest(TempRequest);
end;
```

Note

If the request results in a status code other than HTTP_OK class of status codes (200-299), then the request queue will automatically stop executing requests until the request is retried or cancelled. This is also the case if the OnComplete event handler raises an exception. You can call the ExecuteRequests method to retry the requests from the last request that failed, or call the CancelRequest to cancel the last request that failed and continue with the next queued request.

Cancelling a Server Request

Sometimes it is necessary to cancel a pending server request, and this can be done by calling TServerRequest Cancel method. If you're using a TServerRequestQueue component, then you can call the CancelRequest or CancelAllRequests methods to cancel one or more queued requests.

This page intentionally left blank

Chapter 5
Using Local Storage

5.1 Introduction

Modern browsers provide a local data store to browser applications for storing and retrieving strings by a key. This local data store is normally not very large (typically, around 5-10MB), and the user can customize the maximum available storage size in the browser, so don't rely on using local storage for storing large amounts of data. However, it can be useful for storing user preferences and interim data that needs to be persisted until the data is saved to a web server application.

Warning
Please be very careful about storing sensitive information in local storage. It is not secure, and uses the normal browser cache. On a private machine/device, this may not be an issue. But do not assume that the user is always using a private machine/device, and present them with an option to operate without storing such information.

Elevate Web Builder surfaces the local storage via the TPersistentStorage component class, and automatically creates two instances of the TPersistentStorage class at application startup:

Storage Type	Instance Variable Name
Per-Session	SessionStorage
Local	LocalStorage

The **SessionStorage** and **LocalStorage** variables are declared in the WebComps unit.

The **SessionStorage** instance represents only per-session storage, meaning that once the application has been unloaded, any strings stored in this data store will be permanently deleted.

The **LocalStorage** instance represents browser-wide storage, and persists across instances of the application.

Note
The local data store is segmented by origin, which means that each unique protocol, host, and port has its own data store to use. So, if you loaded your application from http://www.mysite.com/myapp, you would see a different data store than if you loaded your application from https://www.mysite.com/myapp. In contrast, you would see the same data store if you loaded your application from http://www.mysite.com/myapp and http://www.mysite/myotherapp.

5.2 Saving Data To Local Storage

You can use the TPersistentStorage Set method to save a string to a specific key in local storage. The following example shows how to use a form method to store a user's display preferences in local (not per-session) storage so that they are present whenever the application is run:

```
uses WebCore, WebComps;

procedure TForm1.Form1Create(Sender: TObject);
begin
   DisplayPrefs:=TStringList.Create;
end;

procedure TForm1.Form1Destroy(Sender: TObject);
begin
   DisplayPrefs.Free;
   DisplayPrefs:=nil;
end;

procedure TForm1.InitDisplayPrefs;
begin
   with DisplayPrefs do
      begin
      Clear;
      Values['ShowMainMenu']:='True';
      Values['ShowToolBar']:='True';
      end;
end;

procedure TForm1.SaveDisplayPrefs;
begin
   LocalStorage.Set('DisplayPrefs',DisplayPrefs.Text);
end;

procedure TForm1.LoadDisplayPrefs;
begin
   if LocalStorage.Exists('DisplayPrefs') then
      DisplayPrefs.Text:=LocalStorage['DisplayPrefs']
   else
      InitDisplayPrefs;
end;
```

Note
The above code is not complete and is only a cut-down example to illustrate the specific local storage concepts discussed here.

5.3 Loading Data from Local Storage

You can use the TPersistentStorage Exists method to determine if a particular key exists in local storage, and the TPersistentStorage Items property to access a string by its key. The following example shows how to use a form method to check for a user's display preferences in local (not per-session) storage, and then load them if they exist, or initialize them if they don't:

```
uses WebCore, WebComps;

procedure TForm1.Form1Create(Sender: TObject);
begin
   DisplayPrefs:=TStringList.Create;
end;

procedure TForm1.Form1Destroy(Sender: TObject);
begin
   DisplayPrefs.Free;
   DisplayPrefs:=nil;
end;

procedure TForm1.InitDisplayPrefs;
begin
   with DisplayPrefs do
      begin
      Clear;
      Values['ShowMainMenu']:='True';
      Values['ShowToolBar']:='True';
      end;
end;

procedure TForm1.SaveDisplayPrefs;
begin
   LocalStorage.Set('DisplayPrefs',DisplayPrefs.Text);
end;

procedure TForm1.LoadDisplayPrefs;
begin
   if LocalStorage.Exists('DisplayPrefs') then
      DisplayPrefs.Text:=LocalStorage['DisplayPrefs']
   else
      InitDisplayPrefs;
end;
```

Note
The above code is not complete and is only a cut-down example to illustrate the specific local storage concepts discussed here.

5.4 Detecting Local Storage Changes

You can assign an event handler to the TPersistentStorage OnChange event to detect when another session modifies any data saved in local (not per-session) storage. The following example shows how to assign a form method (event handler) to the OnChange event for the global LocalStorage instance of the TPersistentStorage class to detect when any other session modifies a user's display preferences in local (not per-session) storage:

```
uses WebCore, WebComps;

procedure TForm1.StorageChange(Sender: TObject; const Key: String;
                               const NewValue: String; const OldValue: String;

                               const URL: String);
begin
   if (Key='') or (Key='DisplayPrefs') then
      LoadDisplayPrefs;
end;

procedure TForm1.Form1Create(Sender: TObject);
begin
   DisplayPrefs:=TStringList.Create;
   LocalStorage.OnChange:=StorageChange;
end;

procedure TForm1.Form1Destroy(Sender: TObject);
begin
   LocalStorage.OnChange:=nil;
   DisplayPrefs.Free;
   DisplayPrefs:=nil;
end;

procedure TForm1.InitDisplayPrefs;
begin
   with DisplayPrefs do
      begin
      Clear;
      Values['ShowMainMenu']:='True';
      Values['ShowToolBar']:='True';
      end;
end;

procedure TForm1.SaveDisplayPrefs;
begin
   LocalStorage.Set('DisplayPrefs',DisplayPrefs.Text);
end;

procedure TForm1.LoadDisplayPrefs;
begin
   if LocalStorage.Exists('DisplayPrefs') then
      DisplayPrefs.Text:=LocalStorage['DisplayPrefs']
   else
      InitDisplayPrefs;
end;
```

Note
The above code is not complete and is only a cut-down example to illustrate the specific local storage concepts discussed here.

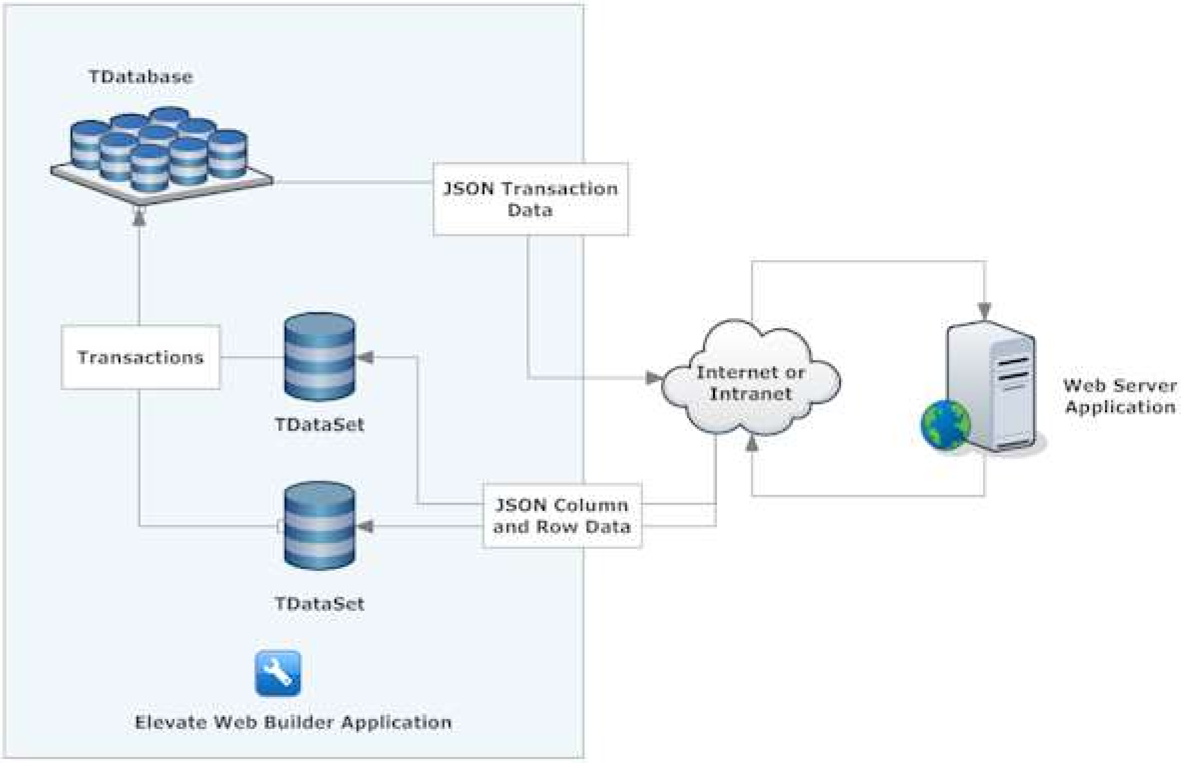
This page intentionally left blank

Chapter 6
Using Databases

6.1 Database Architecture

Elevate Web Builder includes extensive database functionality for easily loading data from the web server and then updating the data on the web server using transactions.

The Elevate Web Builder database functionality has the following architecture:



Elevate Web Builder Database Architecture

The database functionality is virtual and handled in-memory in the Elevate Web Builder client application using a disconnected database architecture. Database access is stateless and all updates to the actual database via the web server are performed optimistically. All database requests/responses use the JSON format for any associated data. Please see the JSON Reference for more information on the schema for the JSON data.

There can be one or more databases (TDatabase instances) in an application, and within each database can be one or more owned datasets (TDataSet instances).

Core Concepts

There are three core concepts in the Elevate Web Builder database functionality:

- **Loading DataSet Columns** - Normally the dataset columns are loaded/defined at design-time in the Elevate Web Builder IDE, but it is possible to dynamically load the columns for a dataset at run-time. The column information comes from the web server application in JSON format and includes basic things such as column name, data type, length, and scale.

- **Loading DataSet Rows** - The dataset rows must be loaded at run-time, and come from the web server application in JSON format. When the rows are loaded, you can specify that the rows be appended to the existing rows in the dataset, or completely replace the current rows in the dataset.

- **Transactions** - By default, transactions are automatically started and committed/rolled back as rows are inserted/saved, updated/saved, and deleted in any datasets contained within a database. Nested transactions are supported, so only the outermost commit operation actually results in communications with the web server application. The automatic transaction handling can be turned off (see the TDatabase component below).

You can find the JSON formats used for all of the above in the JSON Reference topic.

> **Note**
> Elevate Web Builder requires that any table that you wish to update, or any table containing content stored in BLOB columns that you wish to load (such as images), **must** have a primary key defined. Elevate Web Builder uses the primary key to uniquely identify each row.

Core Components

The database functionality contains several core components, all residing in the WebData unit in the Elevate Web Builder component library.

TDatabase
A global TDatabase component instance is auto-created at application startup for both visual and non-visual projects, and is simply called **Database**. This singleton instance of the TDatabase component is used to keep track of all datasets dropped directly on forms and provides methods for iterating over such datasets.

In addition to this default singleton database instance, you can add explicit TDatabase instances to a visual project by dragging and dropping a database defined in the Database Manager into the Project Manager for the currently-opened project. When the database is dropped on or within the **Units** node of the project manager, a new TDatabase (or descendant) instance will be created for the project, along with an associated unit, and all of the defined datasets for the database will automatically be created as TDataSet instances in the new database instance.

The TDatabase AutoTransactions property is used to control whether transactions are automatically handled by the database instances. Please see the Transactions topic for more information on how the AutoTransactions property affects transaction handling.

TDataSet
TDataSet components can either be dropped directly on a form or database at design-time in a visual project, or created at run-time in both visual and non-visual projects. The Columns property contains the column definitions for the dataset.

The columns for a dataset can be defined manually at design-time or load at run-time using the

TDatabase LoadColumns method (via the TDatabase instance that contains the TDataSet instance) or the TDataSet LoadColumns method. The primary difference between the two is that the TDatabase LoadColumns method transparently handles the server request to the web server for retrieving the columns in JSON format, whereas the TDataSet LoadColumns method simply accepts a JSON string containing the columns, and leaves the details of where the JSON originated up to the caller.

Rows must be loaded from the web server application at run-time using the TDatabase LoadRows method (via the TDatabase instance that contains the TDataSet instance) or the TDataSet LoadRows method. The primary difference between the two is that the TDatabase LoadRows method transparently handles the server request to the web server for retrieving the row data in JSON format, whereas the TDataSet LoadRows method simply accepts a JSON string containing the row data, and leaves the details of where the JSON originated up to the caller.

You can navigate the rows in a TDataSet component by using the First, Prior, Next, and Last methods.

The TDataSet component also allows you to Insert, Update, and Delete rows, as well as Find and Sort rows.

6.2 Creating and Using Databases

Before using the TDatabase component, you must first create an instance of the component, which you can do at design-time or at run-time. A global TDatabase instance called **Database** is automatically created at application startup and is used as the default database for any datasets that are created without being specifically associated with a database. Please see the Creating and Loading DataSets topic for more information on how datasets are associated with databases at creation time.

Creating a Database at Design-Time

The easiest way to create a database is by using the Database Manager in the IDE to define a database and its contained datasets. Once a database has been defined under a database in the database manager, you can easily add the database to an existing application by dragging the database from the database manager and dropping it into the project manager for the currently-opened project. When the database is dropped on or within the **Units** node of the project manager, a new TDatabase (or descendant) instance will be created for the project, along with an associated unit, and all of the defined datasets for the database will automatically be created as TDataSet instances in the new database instance.

At design-time, TDatabase instances act (and are stored) like forms but are actually just containers that allow non-visual components like TDataSet instances to be dropped on to the database designer surface. The database designer only allows for non-visual components to be placed on the designer surface, and the visual size of the database instance in the designer is exclusively a design-time property. Please see the Using the Form and Database Designers topic for more information on how to use the database designer.

Authenticating Requests

You can use the TDatabase UserName and Password properties to specify a user name and password to be used with any database requests to the web server. The TDatabase AuthenticationMethod property controls how the authentication information is sent to the web server.

If the AuthenticationMethod property is set to amHeaders (the default), then the user name and password are added as custom headers to the web server request as follows:

```
X-EWBUser: <User Name>
X-EWBPassword: <Password>
```

If the AuthenticationMethod property is set to amParameters, then the user name and password are added as parameters to the web server request as follows:

```
<Database Resource URL>&user=<User Name>&password=<Password>
```

Warning
Elevate Web Builder uses the AJAX functionality in browsers to perform database requests, and this functionality is limited in its ability to perform authentication via native browser methods. Therefore, you should always use secure connections (https) to the web server with any database requests. This is especially true if using the parameter-based authentication, but is also true if you are using datasets with BLOB columns that will require authentication information in their load URL parameters. Please see the JSON Reference topic for more information on BLOB loading.

Database Request Queue

Each TDatabase instance contains a request queue that is used for all database requests to the web server. Elevate Web Builder automatically handles building and sending all databases requests as the database functionality is used in all TDatabase and TDataSet instances. However, if an error occurs during any database request, the request queue is paused and all queued database requests, including the request that failed, are effectively stalled. You can use the TDatabase NumPendingRequests property to determine how many pending requests are present in the request queue, and the TDatabase RetryPendingRequests and CancelPendingRequests methods to retry or cancel any pending requests in the database request queue.

Please see the Executing a Server Request topic for more information on how web server requests are executed.

Transactions

By default, each TDatabase instance automatically handles transactions without requiring them to be manually started/committed/rolled back. This behavior is controlled via the TDatabase AutoTransactions property. Please see the Transactions topic for more information on how database transactions work in Elevate Web Builder.

Database Parameters

You can use the TDatabase Params property to specify database-specific parameters that will be passed as URL parameters with all database requests originating from the database. This is useful for situations where you want to tag all database requests with application-specific information, such as session IDs or tokens. The Params property is a string list (TStrings) of "name=value" pairs that represents the database URL parameters.

6.3 Creating and Loading DataSets

Before using the TDataSet component, you must first create an instance of the component, which you can do at design-time or at run-time.

Creating a DataSet at Design-Time

The easiest way to create a dataset is by using the Database Manager in the IDE to define a database and its contained datasets. Once a dataset has been defined under a database in the database manager, you can easily add the dataset to an existing application by simply dragging it from the database manager and dropping it on a form or database. The relevant property information, including the column definitions, will automatically be populated for the dataset. A database defined in the database manager can be used to create a database in a project by dragging the database from the database manager and dropping it into the project manager for the currently-opened project. When the database is dropped on or within the **Units** node of the project manager, a new TDatabase (or descendant) instance will be created for the project, along with an associated unit, and all of the defined datasets for the database will automatically be created as TDataSet instances in the new database instance.

If you do not wish to use the database manager to create a dataset, you can also create a new dataset by dragging a TDataSet component from the component palette and dropping it on a form or database. Please see the Using the Form and Database Designers topic for more information on the required steps to complete this action. Once you have dropped the TDataSet component on a form or database, you can manually define the columns in the dataset by double-clicking on the TDataSet's Columns property. This will launch the Columns Editor directly under the object inspector, and you can then use the Columns Editor to add, edit, or delete the columns in the dataset.

Creating a DataSet at Run-Time

In cases where visual forms and databases are not being used, such as with a non-visual project or in a library procedure/function, you can create a dataset instance at run-time using code. The following is an example of creating a dataset, opening it, and populating it with some rows at run-time:

```
function CreateStatesDataSet: TDataSet;
begin
   Result:=TDataSet.Create(nil);
   with Result.Columns.Add do
      begin
      Name:='Abbrev';
      DataType:=dtString;
      Length:=2;
      end;
   with Result do
      begin
      Open;
      Insert;
      Columns['Abbrev'].AsString:='CA';
      Save;
      Insert;
      Columns['Abbrev'].AsString:='FL';
      Save;
      Insert;
      Columns['Abbrev'].AsString:='NY';
      Save;
      end;
end;
```

Datasets are associated with a given database by being created with the database as the (sole) owner parameter. As you can see in the above example, the dataset is created with a nil owner parameter, which will cause this dataset instance to be associated with the global **Database** TDatabase instance.

Loading a DataSet at Run-Time

As seen in the above example, you can add rows directly at run-time without ever having to communicate with the web server in order to request data. However, most applications will need to load rows into a dataset from a database by using the web server application as middleware for serving up the necessary rows. There are two different ways to load rows into a dataset at run-time: the TDatabase LoadRows method or the TDataSet LoadRows method.

TDatabase LoadRows Method

The TDatabase LoadRows method is the easiest way to load the rows into a dataset because it automatically handles the actual server request to the web server. The TDatabase component uses the following properties to construct the GET request to the web server for the rows:

- TDatabase BaseURL
  This property defaults to 'databases', but can be changed to any value that you wish. Please note that it is best to use a relative URL path here so that all requests will be made relative to the URL from which the application was loaded. If you're accessing a database module then, by default, you should set this property to 'databasemodules/<module name>', where <module name> is the name of the database module that you wish to access. Please see the Creating Web Server Modules for more information on creating database modules to handle database requests.

- TDatabase DatabaseName
  This property defaults to the same value as the TDatabase component's Name property, but is automatically populated for you if you use the drag-and-drop method of creating a TDatabase at design-time. This property can be changed to any value that you wish, and is simply used to identify the database via a URL parameter used for the web server request.

- TDatabase Params
  This property is a string list (TStrings) of "name=value" pairs that represents the URL parameters for all web server requests for the database. These parameters are strictly application-specific and are not used by by the TDatabase component.

- TDataSet DataSetName
  This property defaults to the same value as the TDataSet component's Name property, but is automatically populated for you if you use the drag-and-drop method of creating a TDataSet at design-time. This property can be changed to any value that you wish, and is simply used to identify the dataset via a URL parameter used for the web server request.

- TDataSet Params
  This property is a string list (TStrings) of "name=value" pairs that represents the URL parameters for the web server request. If the dataset that is being loaded is a query that requires parameters, then you should make sure to specify them using this property.

As an example, consider a database and dataset that is defined as the following in the database manager in the IDE:

```
Database Name: Production

DataSet Name: CustomerOrders

Row Source:

SELECT * FROM custord
WHERE CustomerID={CustomerID='ADF'}

Base Table: custord
```

Assuming that a dataset instance called "CustomerOrders" was created at design-time by dragging and dropping the dataset from the database manager on to a form called "MasterDetailForm", the following code is all that would be needed to load the dataset:

```
procedure TMasterDetailForm.LoadOrders;
begin
   CustomerOrders.Params.Clear;

      CustomerOrders.Params.Add('CustomerID='+QuotedStr(Customer.Columns['CustomerI
      D'].AsString));
   Database.DatabaseName:='Production';  // Uses the default global Database
      TDatabase instance
   Database.LoadRows(CustomerOrders);
end;
```

Note
 You should always use single quotes around all string parameters. Failure to do so will result in the dataset load not working correctly. Use the QuotedStr function to ensure that any string parameters are properly quoted.

In the above example, the relative URL that will be used for the web server GET request would be:

```
databases?method=rows&database=Production&dataset=CustomerOrders&CustomerID='
      ADF'
```

If the application was loaded from 'http://localhost', then the complete URL used for the web server GET request would be:

```
http://localhost/databases?method=rows&database=Production&dataset=CustomerOr
      ders&CustomerID='ADF'
```

If aren't using the global **Database** TDatabase instance and, instead, have created a TDatabase instance in the application, then the code is only slightly different. Assuming that a database instance called "Production" and a dataset instance called "CustomerOrders" was created at design-time by dragging and dropping the database from the database manager on to the project manager, the following code is all that would be needed to load the dataset:

```
procedure TProduction.LoadOrders;
begin
   CustomerOrders.Params.Clear;

      CustomerOrders.Params.Add('CustomerID='+QuotedStr(Customer.Columns['CustomerI
      D'].AsString));
   LoadRows(CustomerOrders);
end;
```

In the above example, the URL used for the web server GET request would be exactly the same as before when the global **Database** TDatabase instance was used instead of a specific TDatabase instance.

After the request is successfully executed, the TDatabase LoadRows method automatically opens the dataset using the TDataSet Open method before also automatically calling the TDataSet LoadRows method.

TDataSet LoadRows Method

The TDataSet LoadRows method directly accepts the dataset rows as a JSON-formatted string. This means that this method is more useful for situations where the dataset rows are stored in memory or local storage and need to be directly loaded from one of those locations. It is recommended that you always use the TDatabase LoadRows method for loading rows from a web server.

Note
The LoadRows method requires that the dataset be open prior to being called. Use the Open method to open the dataset.

Tracking Load Operations

The TDataSet BeforeLoad event is fired before the dataset load actually begins. To prevent the load from occurring, return False as the result in an event handler for this event.

If a dataset load request was sent to the web server and was not successful due to an exception or the web server application returning an HTTP result code other than 200 (OK), the OnLoadError event will be fired and will include the error message. If an event handler is not defined for the OnLoadError event, then an exception will be raised with the error message. If a load fails for any reason, then the load request is placed in a pending requests queue. This is also true for transaction commits. This queue ensures that the database requests can be retried and, when retried, are sent to the web server in the order in which they occurred. You can see if there are any pending database requests by examining the TDatabase NumPendingRequests property. If the NumPendingRequests property is greater than 0, then there are commit and/or dataset load requests that need to be retried at some point. Use the TDatabase RetryPendingRequests method to retry any pending database requests, and the TDatabase CancelPendingRequests method to cancel any pending database requests.

The TDataSet AfterLoad event is fired after the dataset load completes successfully. If there were any errors during the load process, then this event handler will not get called.

6.4 Navigating DataSets

The TDataSet component provides several methods for navigating the rows present in the underlying dataset, as well as properties for obtaining information about the current row position and reading data from the current row.

Moving the Row Pointer

To move the row pointer to a different position in the dataset, use the TDataSet First, Prior, Next, Last, MoveTo, and MoveBy methods. Use the TDataSet BOF, EOF, and RowNo properties to obtain information about the current row position.

The following example navigates from the beginning of a dataset to the end, appending each order ID to a string:

```
var
   OrderIDs: String='';
begin
   with CustomerOrders do
      begin
      First;
      while (not EOF) do
         begin
         if (OrderIDs='') then
            OrderIDs:=Columns['OrderID'].AsString
         else
            OrderIDs:=OrderIDs+', '+Columns['OrderID'].AsString;
         Next;
         end;
      end;
end;
```

Bookmark Operations

Sometimes it is necessary to save the current row pointer, perform some operations that may or may not move the row pointer, and then return to the saved row pointer. The TDataSet SaveBookmark, GotoBookmark, and FreeBookmark methods provide the bookmark functionality for datasets. Bookmarks include a non-volatile row ID and BOF/EOF information so that a row pointer can be restored even when the active sort has been changed. The only case when a row pointer cannot be restored is when the row represented by the bookmark has been deleted.

Note
Bookmarks are automatically pushed and popped from an internal bookmark stack for the dataset, so nested calls to SaveBookmark and GotoBookmark/FreeBookmark will automatically work properly as long as the number of GotoBookmark/FreeBookmark calls matches the number of SaveBookmark calls. Also, GotoBookmark and FreeBookmark are mutually-exclusive: both methods free the active bookmark, but only the GotoBookmark method actually tries to navigate to the active bookmark before freeing it.

The following example saves the current row pointer as a bookmark, updates a column in all of the rows, and then restores the row pointer by calling GotoBookmark:

```
procedure TOrderEntryDlg.UpdateLineNumbers;
begin
   with CustomerItems do
      begin
      DisableControls;
      try
         SaveBookmark;
         try
            First;
            while (not EOF) do
               begin
               Update;
               Columns['LineNo'].AsInteger:=RowNo;
               Save;
               Next;
               end;
         finally
            GotoBookmark;
         end;
      finally
         EnableControls;
      end;
      end;
end;
```

Reading Column Values

The TDataSet Columns property allows you to read the column values for the current row. You can access a column in the Columns property by its index or by its name via the TDataColumns Column property. However, since the Column property is the default property for the TDataColumns object, you can omit it when referencing the Columns property. The following example loops through all columns in a dataset and appends their name to a string:

```
var
   I: Integer;
   ColumnNames: String='';
begin
   with CustomerOrders do
      begin
      for I:=0 to Columns.Count-1 do
         begin
         if (ColumnNames='') then
            ColumnNames:=Columns[I].Name
         else
            ColumnNames:=ColumnNames+','+Columns[I].Name
         end;
      end;
end;
```

Each TDataColumn object present in the TDataSet Columns property has several As* properties that allow you to access the data in the column for the current row as a particular type. Type conversions are performed automatically wherever necessary. However, certain type conversions are impossible and will, if attempted, cause an exception to be raised. For example, the following code will cause an exception to be raised because the OrderDate column, which has a type of dtDate, cannot be converted to a Boolean value:

```
begin
   with CustomerOrders do
      Result:=Columns['OrderDate'].AsBoolean;
end;
```

To determine if a column is Null, you can use the TDataColumn Null property.

Tracking Navigation Operations

The TDataSet BeforeScroll event is fired before the dataset's row pointer moves during navigation. To prevent the navigation from occurring, return False as the result in an event handler for this event.

The TDataSet AfterScroll event is fired after the dataset's row pointer is moved.

The following TDataSet property assignments cause the BeforeScroll and AfterScroll events to be triggered:

RowID
RowNo

The following TDataSet methods cause the BeforeScroll and AfterScroll events to be triggered:

First
Prior
Next
Last
MoveBy
MoveTo
Find
Sort
GotoBookmark

6.5 Searching and Sorting DataSets

The TDataSet component provides several methods for searching and sorting the rows present in the underlying dataset, as well as properties for obtaining information about the active sort.

Sorting the Rows

To sort the rows in a dataset, use the Sort method. To specify the columns to sort, assign the desired value to the TDataColumn SortDirection property in the order that reflects the column order of the desired sort. Use the TDataSet SortCaseInsensitive property to specify that the sort should be case-insensitive, and the SortLocaleInsensitive property to specify that the sort should be locale-insensitive. The default value for both properties is False.

The following example sorts the Products dataset based upon descending list price:

```
begin
   with Products do
      begin
      Columns['ListPrice'].SortDirection:=sdDescending;
      Sort;
      end;
end;
```

Once a sort has been established, the TDataSet Sorted will return True and the dataset will automatically keep the rows sorted accordingly as rows are inserted, updated, or deleted. The TDataColumn SortIndex property can be examined to determine where a column resides in the active sort. To clear an existing sort, simply assign a value of sdNone to the SortDirection property of all sorted columns.

Searching for a Row

The TDataSet InitFind and Find methods allow you to search the rows in the dataset for a particular set of column values. The first step to executing a search is to call the InitFind method, which puts the dataset in the "Find" state, which is represented by the TDataSet State property. Once the dataset is in the "Find" state, you can assign values to the columns in the dataset and then call the Find method to execute the actual search. If there is a sort active on the dataset, then it will be used for satisfying the Find operation if the modified columns and the CaseInsensitive parameter to the Find method match the active sort. For example, the following example sorts the Products dataset by the ProductID column and then executes a case-insensitive Find operation on the ProductID column for the 'PEN-BP-12PK' product ID:

```
begin
   with Products do
      begin
      Columns['ProductID'].SortDirection:=sdAscending;
      SortCaseInsensitive:=True;
      Sort;
      InitFind;
      Columns['ProductID'].AsString:='PEN-BP-12PK';
      if Find(False,True) then
         Result:=True
      else
```

```
         Result:=False;
      end;
end;
```

To perform a search for the row with the column values that are nearest to the specified Find values, simply pass True as the first parameter to the Find method.

Note
The TDataSet component determines which columns participate in the Find operation, and subsequently which columns need to match the active sort, based upon which columns have been modified since the InitFind method was called. In order to perform a nearest value search, the modified columns and the CaseInsensitive Find parameter (the second) must match the active sort.

6.6 Updating DataSets

The TDataSet component provides several methods for inserting, updating, and deleting rows in the underlying dataset, as well as properties for reading both the current and old column values from the current row.

Inserting New Rows

Inserting a new row in a dataset is a three-step process. First, use the TDataSet Insert method to put the dataset into the insert state. This will:

- Fire the BeforeInsert event handler, if one is defined. To prevent the insert from occurring, return False as the result in the event handler.

- If the OwnerDatabase's AutoTransactions property is True (the default), then start a new transaction and then create a new row. If the Append flag (default False) is passed to the Insert method, then the new row will be appended to the end of the dataset, otherwise the new row will be inserted at the current row pointer in the dataset.

- Fire the OnInitrow event handler, if one is defined. The OnInitRow event handler allows the application to assign values to the columns in the new row without causing any of the columns, or the row, to be flagged as modified. This is a good place to assign default values for columns.

- Change the TDataSet State property to dsInsert once the dataset is in the insert state.

- Fire the AfterInsert event handler, if one is defined.

Once the dataset is in the insert state, you can use the Columns property to read or assign new values to the various columns in the row using the TDataColumn As* properties. When a column is assigned a new value, its Modified property is set to True.

When all column modifications have been made, use the Save method to complete the insert. This will:

- Fire the BeforeSave event handler, if one is defined. To prevent the save from occurring, return False as the result in the event handler.

- Save the row in the dataset, logging the insert if a transaction is in progress. At this point, the Modified property for the columns in the row will be reset to False.

- Fire the AfterSave event handler, if one is defined.

- Update any active sort and change the TDataSet State property to dsBrowse to reflect that the dataset is now in the browse state.

> **Note**
> With no active sort, rows are always sorted by their actual insertion order, so even if the Insert method was called without the Append flag, the newly-inserted row will move to the end of the dataset after the Save method completes.

- If the OwnerDatabase's AutoTransactions property is True (the default), then commit the active transaction.

If you want to cancel the insert operation, you can use the TDataSet Cancel method. This will:

- Fire the BeforeCancel event handler, if one is defined. To prevent the cancel from occurring, return False as the result in the event handler.
- Discard the row. The row pointer will return to the row pointer that was active prior to the Insert method being called.
- Fire the AfterCancel event handler, if one is defined.
- Change the TDataSet State property to dsBrowse to reflect that the dataset is now in the browse state.
- If the OwnerDatabase's AutoTransactions property is True (the default), then roll back the active transaction.

The following example inserts a new product into the Products dataset:

```
begin
   with Products do
      begin
      Products.Insert; // Required to avoid conflict with Insert system
      function
      Columns['ProductID'].AsString:='PHONE-HEADSET';
      Columns['Description'].AsString:='Hands-free phone handset';
      Columns['ListPrice'].AsFloat:=15.00;
      Columns['Shipping'].AsFloat:=2.00;
      Save;
      end;
end;
```

Updating Existing Rows

Updating an existing row in a dataset is a three-step process. First, use the TDataSet Update method to put the dataset into the update state. This will:

- Fire the BeforeUpdate event handler, if one is defined. To prevent the update from occurring, return False as the result in the event handler.

- If the OwnerDatabase's AutoTransactions property is True (the default), then start a new transaction.
- Change the TDataSet State property to dsUpdate once the dataset is in the update state.
- Fire the AfterUpdate event handler, if one is defined.

Once the dataset is in the update state, you can use the Columns property to read or assign new values to the various columns in the row using the TDataColumn As* properties. When a column is assigned a new value, its Modified property is set to True. You can use the TDataColumn OldValue property to access the value of any column before any new assignments were made to the row.

When all column modifications have been made, use the Save method to complete the update. This will:

- Fire the BeforeSave event handler, if one is defined. To prevent the save from occurring, return False as the result in the event handler.
- Save the row in the dataset, logging the update if a transaction is in progress. At this point, the Modified property for the columns in the row will be reset to False.
- Fire the AfterSave event handler, if one is defined.
- Update any active sort and change the TDataSet State property to dsBrowse to reflect that the dataset is now in the browse state.

Note
If any of the columns modified during the update are part of the active sort, then the row will automatically move to the correct position in the active sort.

If you want to cancel the update operation, you can use the TDataSet Cancel method. This will:

- Fire the BeforeCancel event handler, if one is defined. To prevent the cancel from occurring, return False as the result in the event handler.
- Discard any modifications to the row. The row pointer will stay in the same location.
- Fire the AfterCancel event handler, if one is defined.
- Change the TDataSet State property to dsBrowse to reflect that the dataset is now in the browse state.
- If the OwnerDatabase's AutoTransactions property is True (the default), then roll back the active transaction.

The following example finds a product in the Products dataset and updates its shipping cost:

```
begin
   with Products do
      begin
      InitFind;
      Columns['ProductID'].AsString:='PHONE-HEADSET';
      if Find(False,True) then
         begin
         Update;
         Columns['Shipping'].AsFloat:=1.80;
         Save;
         end;
      end;
end;
```

Deleting Existing Rows

Deleting an existing row in a dataset can be done by calling the TDataSet Delete method. This will:

- Fire the BeforeDelete event handler, if one is defined. To prevent the delete from occurring, return False as the result in the event handler.

- If the OwnerDatabase's AutoTransactions property is True (the default), then start a new transaction and delete the existing row.

- Fire the AfterDelete event handler, if one is defined.

The following example finds a product in the Products dataset and deletes it:

```
begin
   with Products do
      begin
      InitFind;
      Columns['ProductID'].AsString:='FLASH-USB-16GB';
      if Find(False,True) then
         Delete;
      end;
end;
```

6.7 Transactions

While datasets can be updated without using transactions, doing so causes all updates to be completely bound to the dataset in the Elevate Web Builder application and unable to ever leave that context (apart from being saved to local storage). Using transactions allows all updates to a dataset to be logged so that the updates can then be sent to the web server application and reflected in an actual database. This is especially important when a dataset is being loaded from a table or query result set present in a database accessible from the web server application.

All transaction properties, methods, and events are contained with the TDatabase component in the WebData unit. The AutoTransactions property is True, by default, and controls whether transactions are automatically started and committed/rolled back as the datasets are updated. Please see the Updating DataSets topic to see how the automatic transactions interact with the various dataset insert, update, and delete operations. The methods for starting, committing, and rolling back transactions are the StartTransaction, Commit, and Rollback methods. Transactions in Elevate Web Builder can be nested, so the TDatabase InTransaction and TransactionLevel properties reflect whether a transaction is active and at what level, respectively. If the TransactionLevel property is -1, then no transactions are active.

Starting a Transaction

Use the TDatabase StartTransaction method to start a transaction. This will increment the current transaction level. All row inserts, updates, and deletes taking place in any owned datasets will be automatically logged as part of the current transaction.

> **Warning**
> If you attempt to close a dataset using the TDataSet Close method while there are operations logged for the current transaction, an exception will be raised.

Committing a Transaction

Use the Commit method to commit the current transaction. This will:

- Fire the BeforeCommit event handler, if one is defined. To prevent the commit from occurring, return False as the result in the event handler.

- If the current transaction level, as reflected by the TDatabase TransactionLevel property, is 0, create a web server POST request and send it to the web server application, with the transaction data included as the POST content in JSON format. To see the JSON format used for the transactions, please see the JSON Reference topic.

- If the current transaction level is greater than 0, then append all operations in the current transaction to the next lower transaction.

- Decrement the current transaction level. If the transaction level is -1, then the TDatabase InTransaction property is set to False.

- If a POST request was not sent to the web server because the current transaction being committed was nested, then immediately fire the AfterCommit event handler, if one is defined.

- If a POST request was sent to the web server and was not successful due to an exception or the web server application returning an HTTP result code other than 200 (OK), the OnCommitError event will be fired and will include the error message. If an event handler is not defined for the OnCommitError event, then an exception will be raised with the error message. If a commit fails for any reason, then the transaction being committed is placed in a pending requests queue. This is also true for general database requests such as load requests. This queue ensures that the database requests can be retried and, when retried, are sent to the web server in the order in which they occurred. You can see if there are any pending database requests by examining the TDatabase NumPendingRequests property. If the NumPendingRequests property is greater than 0, then there are commit and/or dataset load requests that need to be retried at some point. Use the TDatabase RetryPendingRequests method to retry any pending database requests, and the TDatabase CancelPendingRequests method to cancel any pending database requests.

- If a POST request was sent to the web server and was successful, then fire the AfterCommit event handler, if one is defined.

Commit POST Requests

The TDatabase component uses the following properties to construct the POST request to the web server when committing a transaction:

- TDatabase BaseURL
  This property defaults to 'datasets', but can be changed to any value that you wish. Please note that it is best to use a relative URL path here so that all requests will be made relative to the URL from which the application was loaded.

- TDatabase DatabaseName
  This property defaults to the same value as the TDatabase component's Name property, but is automatically populated for you if you use the drag-and-drop method of creating a TDatabase at design-time. This property can be changed to any value that you wish, and is simply used to identify the database via a URL parameter used for the web server request.

- TDatabase Params
  This property is a string list (TStrings) of "name=value" pairs that represents the URL parameters for all web server requests for the database. These parameters are strictly application-specific and are not used by by the TDatabase component.

As an example, consider a typical transaction commit. In such a case, the relative URL that will be used for the web server POST request would be:

```
databases?method=commit&database=Production
```

If the application was loaded from 'http://localhost', then the complete URL used for the web server POST request would be:

```
http://localhost/databases?method=commit&database=Production
```

Now consider a transaction commit where the BaseURL property is set to 'databases/transact.php'. In such a case, the relative URL that will be used for the web server POST request would be:

```
databases/transact.php?method=commit&database=Production
```

If the application was loaded from 'http://localhost', then the complete URL used for the web server POST request would be:

```
http://localhost/databases/transact.php?method=commit&database=Production
```

Rolling Back a Transaction

Use the Rollback method to roll back the current transaction. This will:

- Fire the BeforeRollback event handler, if one is defined. To prevent the rollback from occurring, return False as the result in the event handler.

- Undo all operations that have taken place in the current transaction. If there are any exceptions during this process, the OnRollbackError event will be fired and will include the error message. If an event handler is not defined for the OnRollbackError event, then an exception will be raised with the error message.

> **Note**
> It is highly unlikely that an exception will ever be raised when a transaction is being rolled back, but it is possible that a catastrophic browser error could cause such an exception.

- Decrement the current transaction level. If the transaction level is -1, then the TDatabase InTransaction property is set to False.

- Fire the AfterRollback event handler, if one is defined.

The following example starts a transaction, deletes all rows, and then commits the transaction.:

```
begin
   Database.StartTransaction;
   with Products do
      begin
      while (RowCount > 0) do
         Delete;
      end;
   Database.Commit;
end;
```

Note
If you attempt to call the TDatabase Commit or Rollback methods when there are no active transactions (InTransaction property is False), then an exception will be raised.

6.8 Responding to DataSet Changes

It is important that one be able to respond to various changes to dataset columns and rows, both for updating control states and for implementing concepts such as master-detail linkages. The TDataSet OnStateChange event is used to track when the TDataSet State changes. The TDataSet OnRowChanged event is used to track when the active row in the dataset changes, or when a column in the active row changes.

State Changes

Define an event handler for the TDataSet OnStateChange event in order to track when the dataset state changes. The following list details the TDataSet methods that cause the dataset state to change, along with the state after the method completes:

Method	After State
Open	dsBrowse
Close	dsClosed
CheckBrowseMode	dsBrowse (if result is True)
InitFind	dsFind
Find	dsBrowse
Insert	dsInsert
Update	dsUpdate
Save	dsBrowse
Cancel	dsBrowse

The following example shows an OnStateChange event handler that displays the state of a dataset called "Vendors" in a label on the current form:

```
procedure TMyForm.VendorsStateChange(Sender: TObject);
begin
   case Vendors.State of
      dsClosed:
         VendorStateLabel.Caption:='Closed';
      dsBrowse:
         VendorStateLabel.Caption:='Browse';
      dsInsert:
         VendorStateLabel.Caption:='Insert';
      dsUpdate:
         VendorStateLabel.Caption:='Update';
      dsFind:
         VendorStateLabel.Caption:='Find';
      end;
end;
```

Row Changes

Define an event handler for the TDataSet OnRowChanged event in order to track when the active row in the dataset changes. The OnRowChanged event is fired when any column in the active row is changed due to a modification, or when the active row changes due to the row pointer moving. If the OnRowChanged event was fired in response to a column modification, then the Column parameter in the event handler will contain an instance of the TDataColumn that was modified. If the OnRowChanged event was fired in response to the entire active row changing, then the Column parameter will be nil.

The following TDataColumn properties and methods will cause the OnRowChanged event to fire with a non-nil Column parameter:

AsString
AsBoolean
AsInteger
AsFloat
AsDate
AsTime
AsDateTime
Clear

The following TDataSet properties and methods will cause the OnRowChanged event to fire with a nil Column parameter:

RowID
LoadRows
Sort
EnableControls
First
Prior
Next
Last
MoveBy
MoveTo
GotoBookmark
InitFind
Find
Insert
Save
Cancel
Delete

The following TDatabase properties and methods will cause the OnRowChanged event to fire with a nil Column parameter:

LoadRows
Rollback

Responding to row changes is important for updating related controls in the user interface. The following example shows an OnRowChanged event handler that responds to row changes by calling methods that update both buttons and labels:

```
procedure TMasterDetailForm.CustomerOrdersRowChanged(Sender: TObject;
                                                       Column: TDataColumn);
begin
   if (Column=nil) then
      begin
      UpdateOrderButtons;
      UpdateOrderLabels;
      end;
end;
```

Responding to row changes is also important for concepts such as master-detail links. The following example shows an OnRowChanged event handler that responds to row changes for loading a detail dataset as the master row changes:

```
procedure TMasterDetailForm.LoadOrders;
begin
   CustomerOrders.Params.Clear;
   CustomerOrders.Params.Add('CustomerID='''+
                             Customer.Columns['CustomerID'].AsString+
                             '''');
   Database.LoadRows(CustomerOrders);
end;

procedure TMasterDetailForm.CustomerRowChanged(Sender: TObject;
                                                 Column: TDataColumn);
begin
   if (Column=nil) then
      begin
      UpdateCustomerButtons;
      LoadOrders;
      end;
end;
```

6.9 Binding Controls to DataSets

The controls in the Elevate Web Builder component library can be used in both an unbound and bound fashion. A control is considered bound when it is explicitly attached to a dataset and one or more columns in the dataset. Most controls bind to a specific dataset column, while certain controls like the TGrid control can bind to multiple dataset columns.

Once a control is bound to a dataset, it will automatically update its contents in response to changes in the dataset. For example, if you insert a new row in the dataset using the TDataSet Insert method, then the control will automatically repopulate with the value from the new row.

Binding to a DataSet

To bind a control to a specific dataset, assign an existing TDataSet instance to the DataSet property of the control. This can be done at design-time or run-time.

> **Note**
> If the TDataSet instance that is assigned to the DataSet property is deleted, the control will automatically assign a value of nil to the DataSet property and the control will become unbound.

However, simply assigning the DataSet property is insufficient for binding a control to a dataset - you must also specify which column in the dataset to bind to. For most controls, this is done by assigning a column name to the DataColumn property. For the TGrid component, you must assign a column name to the DataColumn property of each TGridColumn in the grid that you wish to be bound to the dataset.

> **Note**
> The TGrid component allows you to to mix bound and un-bound columns within the same grid control.

Auto-Editing of Bound Controls

The value of the TDataSet AutoEdit property determines whether modifications to the contents of bound controls are allowed when the dataset is not in an editable state (TDataSet State property is dsInsert or dsUpdate). If the AutoEdit property is True, then any modification to a bound control will cause the attached dataset to insert a new row if the dataset is empty, or begin updating the current row if the dataset is not empty. If the AutoEdit property is False, then bound controls are effectively read-only until either a new row is inserted or an existing row is updated in the dataset.

> **Note**
> The TGrid control has two properties that can still enable a user to insert or delete rows in a bound grid. They are the AllowInserts and AllowDeletes properties, respectively. Be sure to set these properties to False if you do not want to allow a user to automatically insert or delete rows by using keystrokes in the grid.

Auto-Editing and Read-Only Columns

Dataset columns can be defined as calculated or read-only using the TDataColumn Calculated and ReadOnly properties, respectively. Any controls bound to a calculated or read-only column will not be editable, and will behave as though the dataset's AutoEdit property is set to False.

6.10 Calculated Columns

As discussed in the Creating and Loading DataSets topic, dataset columns are normally defined automatically when dragging a dataset from the IDE's Database Manager and dropping the dataset on a form or database designer, or they can be loaded at runtime from the web server via the TDataSet LoadColumns method.

However, in some cases you may want to define columns that derive their contents from a calculation. In Elevate Web Builder, these are, of course, called calculated columns. Creating a calculated column is very simple:

- Create the column as you normally would, using the Add method of the TDataSet Columns.
- Set the new column's Calculated property to True.
- Define an event handler for the TDataSet OnCalculateRow event that executes the calculation code and assigns a value to the new calculated column.

Whenever a column in a row is updated, the OnCalculateRow event handler will be triggered so that any calculated columns can be re-computed for that row.

Any editable controls bound to a calculated column will automatically be read-only.

The following is an example of creating a calculated column that shows concatenated information from two other columns in the dataset:

```
procedure TForm1.Form1Create(Sender: TObject);
begin
   with Albums.Columns.Add do
      begin
      Name:='ArtistYear';
      DataType:=dtString;
      Length:=60;
      Calculated:=True;
      end;
   Albums.OnCalculateRow:=AlbumsCalculateRow;
end;

procedure TForm1.AlbumsCalculateRow(Sender: TObject; Column: TDataColumn);
begin
   Albums.Columns['ArtistYear'].AsString:=Albums.Columns['Artist'].AsString+
            ' ('+Albums.Columns['Year'].AsString+')';
end;
```

Note
Do not attempt to programmatically modify a calculated column outside of an OnCalculateRow event handler. Attempting to do so will result in an error. Also, you **cannot** modify non-calculated columns in an OnCalculateRow event handler.

6.11 API Reference

Elevate Web Builder uses a defined server request API for handling database operations between an application and the web server. Both the internal web server in the IDE and the included external Elevate Web Builder Web Server include support for this API. However, for other web servers the API support must be coded via a layer in the web server application, whether it is coded using PHP, Ruby, ASP.NET, or any other type of web server language or scripting environment. This reference will assist you in building such a layer in your web server application.

Elevate Web Builder uses three types of API calls for the database functionality:

- DataSet columns
- DataSet rows
- Transactions

DataSet Columns

Dataset columns are requested from the web server using an HTTP GET request when the TDataSet LoadColumns method is called from the application. The GET request URL will have the following format:

```
<Database Resource Name>?method=columns&database=<Database
      Name>&dataset=<DataSet Name>[<Custom Parameters>]
```

where <Database Resource Name> is the base resource name for the database API (the default is 'databases'), <Database Name> is the name of the database, <DataSet Name> is the name of the dataset, and <Custom Parameters> are any additional custom parameters sent along with the base parameters. If the client application is using URL authentication parameters (the default is to use HTTP headers), then there may be additional user and password parameters/values included in the complete URL. Please see the Creating and Using Databases topic for more information on specifying the authentication method for database requests.

The response from a dataset columns request will indicate an HTTP status code of 200 (OK) along with the JSON column data as the included response content, or a non-200 error status code.

Please see the JSON Reference topic for more information on the structure of the JSON column data returned.

DataSet Rows

Dataset rows are requested from the web server using an HTTP GET request when the TDataSet LoadRows or the TDatabase LoadRows method is called from the application. The GET request URL will have the following format:

```
<Database Resource Name>?method=rows&database=<Database
      Name>&dataset=<DataSet Name>[<Custom Parameters>]
```

where <Database Resource Name> is the base resource name for the database API (the default is 'databases'), <Database Name> is the name of the database, <DataSet Name> is the name of the dataset, and <Custom Parameters> are any additional custom parameters sent along with the base parameters. If the client application is using URL authentication parameters (the default is to use HTTP headers), then there may be additional user and password parameters/values included in the complete URL. Please see the Creating and Using Databases topic for more information on specifying the authentication method for database requests.

The response from a dataset rows request will indicate an HTTP status code of 200 (OK) along with the JSON row data as the included response content, or a non-200 error status code.

Please see the JSON Reference topic for more information on the structure of the JSON row data returned.

Transactions

Transaction operations are sent to the web server using an HTTP POST request when the TDatabase Commit method is called from the application, and the current TransactionLevel is 0. The POST request URL will have the following format:

```
<Database Resource Name>?method=commit&database=<Database Name>[<Custom
      Parameters>]
```

where <Database Resource Name> is the base resource name for the database API (the default is 'databases'), <Database Name> is the name of the database, and <Custom Parameters> are any additional custom parameters sent along with the base parameters. If the client application is using URL authentication parameters (the default is to use HTTP headers), then there may be additional user and password parameters/values included in the complete URL. Please see the Creating and Using Databases topic for more information on specifying the authentication method for database requests.

The response from a database commit request will indicate an HTTP status code of 200 (OK) or a non-200 error status code.

Please see the JSON Reference topic for more information on the structure of the JSON transaction data that should be included in the request content.

6.12 JSON Reference

Elevate Web Builder uses the JSON (JavaScript Object Notation) format for handling database operations between an application and the web server. Both the internal web server in the IDE and the included external Elevate Web Builder Web Server include support for providing JSON column and row data for any datasets in databases defined in the Database Manager, as well as accepting transactional JSON data for inserts, updates, and deletes. However, for other web servers the JSON must be generated and consumed via a layer in the web server application, whether it is coded using PHP, Ruby, ASP.NET, or any other type of web server language or scripting environment. This reference will assist you in building such a layer in your web server application.

For more general information on JSON, please see the following link:

JSON Reference

Elevate Web Builder uses three types of JSON formats for the database functionality:

- DataSet columns
- DataSet rows
- Transactions

DataSet Columns

Dataset columns are requested from the web server using an HTTP GET request when the TDataSet LoadColumns method is called from the application. The JSON returned by the web server should have the following format:

```
{ columns: [ <Column>, <Column>, <Column>, ... ] }

(... denotes more columns)

<Column> = { name: <Name>, type: <Type>, length: <Length>, scale: <Scale>}

<Name> = String (Example: "Customer No")

<Type> = Integer with a value of 0 through 8 (see below)

<Length> = Integer or null (Example: 20)

<Scale> = Integer or null (Example: 2)
```

Column Types

The following details the various column types and how they should be specified:

Column Type	Description
0	Unknown type - will cause an error when the columns are loaded
1	String - requires a column length for fixed-length columns, null for variable-length columns
2	Boolean
3	Integer
4	Float - can have a column scale specified
5	Date
6	Time
7	Date/Time
8	BLOB

BLOB Column Types

In Elevate Web Builder datasets, BLOB columns are handled as String columns with a null length, and usually contain a URL that is used to dynamically load the BLOB data into a TImage, TAudio, or TVideo control. However, if a BLOB column is actually a CLOB (Character Large Object) column, then it will/should be defined and handled as an actual string, and not a URL.

Elevate Web Builder also supports the use of an additional String column for BLOB columns that indicates the MIME type of the BLOB column data. Such a column should be named:

```
<BLOB Column Name>_ContentType
```

If Elevate Web Builder finds a column with this name, it will use the contents of the column as the response Content-Type header when returning the BLOB data for BLOB column load requests. This is especially necessary for binary formats that cannot be detected by the browser automatically.

See the BLOB Column Data section below for more information on handling BLOB column data.

Example JSON

The following is an example of the JSON for a products table:

```
{ "columns": [
{ "name": "ProductID","type": 1,"length": 30,"scale": null },
{ "name": "Description","type": 1,"length": 60,"scale": null },
{ "name": "ListPrice","type": 4,"length": null,"scale": 2 },
{ "name": "Shipping","type": 4,"length": null,"scale": 2 }
] }
```

DataSet Rows

Dataset rows are requested from the web server using an HTTP GET request when the TDataSet LoadRows method or the TDatabase LoadRows method is called from the application. The JSON returned by the web server should have the following format:

```
{ rows: [ <Row>, <Row>, <Row>, ... ] }

(... denotes more rows)

<Row> = { <Column Name>: <Column Data>, <Column Name>: <Column Data>, ... }

(... denotes more column data)

<Column Name> = String (Example: "Customer No")

<Column Data> = Valid column data or null (see below)
```

Column Data

The following details the various column types and how the column data should be formatted for each:

Column Type	Description
String BLOB	Enclose non-**null** values in double quotes.
Boolean	Specify **true** or **false** literals for non-**null** values.
Integer	Specify any valid integer value (positive or negative) for non-**null** values.
Float	Specify any valid floating-point value for non-**null** values. If not **null**, the incoming value must use the period (.) decimal separator if it contains fractional digits.
Date Time Date/Time	Specify any valid integer value (positive or negative) for non-**null** values. If not **null**, the incoming value represents the number of milliseconds since midnight on January 1, 1970, and can be negative for time values

BLOB Column Data

As mentioned above in the BLOB Column Types section, BLOB columns that are actually binary and not CLOB columns will/should be sent to the Elevate Web Builder application from the web server application as URLs that provide a link to the BLOB data. These links will be passed back to the web server application from the Elevate Web Builder application unchanged, so they can also contain information such as authentication. By default, the internal web server in the IDE, as well as the Elevate Web Builder Web Server, generate the URLs in the following format:

```
?method=load&database=<Database Name>&dataset=<DataSet Name>&column=<Column
      Name>&row=<Primary Key Values>[&user=<User Name>&password=<Password>]
```

The user and password parameters are only included when the original dataset rows request was authenticated. Except for public data, one should **always** use authentication for database requests. For more information, please see the Creating and Loading DataSets topic.

> **Warning**
> Elevate Web Builder uses the AJAX functionality in browsers to perform database requests, and this functionality is limited in its ability to perform authentication via native browser methods. Therefore, you should always use secure connections (https) to the web server with any database requests. This is especially true if using BLOB columns that will require authentication information in their URL parameters.

Example JSON

The following is an example of the JSON for a products table:

```
{ "rows": [
{ "ProductID": "9V-BATTERY-12PK",
  "Description": "12-pack of 9-volt batteries",
  "ListPrice": 20, "Shipping": 2 },
{ "ProductID": "9V-BATTERY-4PK",
  "Description": "4-pack of 9-volt batteries",
  "ListPrice": 4.5, "Shipping": 1.5 },
{ "ProductID": "CALCULATOR-BUSINESS",
  "Description": "Business calculator",
  "ListPrice": 10, "Shipping": 1 },
{ "ProductID": "CASH-REGISTER",
  "Description": "Cash register with thermal printer",
  "ListPrice": 170, "Shipping": 10 },
{ "ProductID": "FLASH-USB-16GB",
  "Description": "16GB USB flash drive",
  "ListPrice": 15, "Shipping": 0.5 },
{ "ProductID": "FLASH-USB-32GB",
  "Description": "32GB USB flash drive",
  "ListPrice": 25, "Shipping": 0.5 },
{ "ProductID": "FLASH-USB-8GB",
  "Description": "8GB USB flash drive",
  "ListPrice": 10, "Shipping": 0.5 },
{ "ProductID": "LABEL-MAKER",
  "Description": "Label maker - plastic labels",
  "ListPrice": 35, "Shipping": 2 },
{ "ProductID": "PEN-BP-12PK",
  "Description": "12-pack of ballpoint pens",
  "ListPrice": 12, "Shipping": 0.6 },
{ "ProductID": "PHONE-HEADSET",
  "Description": "Hands-free phone headset",
  "ListPrice": 15, "Shipping": 2 },
{ "ProductID": "PHONE-SYSTEM-4HS",
  "Description": "4-handset phone system with main base",
  "ListPrice": 120, "Shipping": 4 },
{ "ProductID": "PROJECTOR-HD",
  "Description": "1080p HD Projector",
  "ListPrice": 850, "Shipping": 56 },
{ "ProductID": "SCANNER-SF",
  "Description": "Sheet-feed paper scanner",
  "ListPrice": 150, "Shipping": 7 },
{ "ProductID": "SHREDDER-SF-CC",
  "Description": "Sheet-feed, cross-cut shredder with bin",
  "ListPrice": 8, "Shipping": 10 },
{ "ProductID": "USB-CARD-READER",
  "Description": "USB magnetic strip card reader",
  "ListPrice": 25, "Shipping": 2 }
] }
```

Transactions

Transaction operations are sent to the web server using an HTTP POST request when the TDatabase Commit method is called from the application, and the current TransactionLevel is 0. The JSON sent to the web server will have the following format:

```
{ operations: [ <Operation>, <Operation>, <Operation>, ... ] }

(... denotes more operations)

<Operation> = { dataset: <DataSet Name>, operation: <Operation Type>,
                beforerow: <Row>, afterrow: <Row> }

<DataSet Name> = String (Example: "Customers")

<Operation Type> = Integer with a value of 0 through 3 (see below)

<Row> = null or { <Column Name>: <Column Data>,
                  <Column Name>: <Column Data>, ... }

(... denotes more column data)

<Column Name> = String (Example: "Customer No")

<Column Data> = Valid column data or null (see above)
```

Operation Types

The following details the various operation types and how the row data will be formatted for each:

Operation Type	Description
1	Insert - the beforerow value will be **null** and the afterrow value will contain the row data for the inserted row.
2	Update - the beforerow value will contain the row data for the row before the update, and the afterrow value will contain the row data for the row after the update (modified values only).
3	Delete - the beforerow value will contain the row data for the row before the deletion, and the afterrow value will be **null**.

Example JSON

The following is an example of the transactional JSON for order and items tables:

```
{ "operations": [
{ "dataset": "CustomerOrders",
  "operation": 1,
  "beforerow": null,
  "afterrow": { "CustomerID": "DM", "OrderID": "DM-201275-134324404",
                "OrderDate": 1341460800000,
                "PONumber": null, "Terms": "Net 30",
                "ShippingTotal": 0.00, "PurchaseTotal": 0.00,
                "OrderTotal": 0.00, "AmountPaid": 0.00, "BalanceDue": 0.00,
                "SpecialInstructions": null }
},
{ "dataset": "CustomerItems",
  "operation": 1,
  "beforerow": null,
  "afterrow": { "OrderID": "DM-201275-134324404", "LineNo": 1,
                "ProductID": "SCANNER-SF", "Quantity": 1,
                "PurchasePrice": 150.00, "Shipping": 7.00,
                "PurchaseTotal": 150.00, "ShippingTotal": 7.00 }
},
{ "dataset": "CustomerItems",
  "operation": 1,
  "beforerow": null,
  "afterrow": { "OrderID": "DM-201275-134324404", "LineNo": 2,
                "ProductID": "FLASH-USB-32GB", "Quantity": 10,
                "PurchasePrice": 25.00, "Shipping": 0.50,
                "PurchaseTotal": 250.00, "ShippingTotal": 5.00 }
},
{ "dataset": "CustomerOrders",
  "operation": 2,
  "beforerow": { "CustomerID": "DM", "OrderID": "DM-201275-134324404",
                 "OrderDate": 1341460800000,
                 "PONumber": null, "Terms": "Net 30",
                 "ShippingTotal": 0.00, "PurchaseTotal": 0.00,
                 "OrderTotal": 0.00, "AmountPaid": 0.00, "BalanceDue": 0.00,
                 "SpecialInstructions": null },
  "afterrow": { "PONumber": "210054", "ShippingTotal": 12.00,
                "PurchaseTotal": 400.00,"OrderTotal": 412.00,
                "BalanceDue": 412.00 }
}
] }
```

This page intentionally left blank

Chapter 7
Using the Web Server

7.1 Starting the Web Server

Elevate Web Builder includes an external, deployable web server along with the IDE and compiler. The Elevate Web Builder Web Server runs on Windows XP or higher as a normal application or Windows service, and automatically supports Elevate Web Builder application database requests. In addition, you can use Embarcadero RAD Studio and Delphi to create native server modules that can be added to the web server in order to handle requests from an Elevate Web Builder application (or any web browser or web browser application).

The web server executable is called ewbsrvr.exe and can found in the \bin\ewbsrvr sub-directory under the main installation directory.

Installing the Web Server as a Service

If you wish to run the web server as a Windows service you must install it as a service by running the web server with the /install command-line switch set. For example, to install the web server as a service using the Run command window under Windows you would specify the following command:

```
ewbsrvr.exe /install
```

To uninstall the web server as a Windows service you must run the web server with the /uninstall command-line switch set. For example, to uninstall the web server as a service using the Run command window under Windows you would specify the following command:

```
ewbsrvr.exe /uninstall
```

If you wish to install the web server so that it does not interact with the desktop at all, which is required in instances where the current user will be logged out of the system, then you should use the /nointeract flag along with the /install command-line switch:

```
ewbsrvr.exe /install /nointeract
```

This will install the service as a non-interactive service and the web server will not display a user interface when it is started.

Finally, by default the service will display a "Service installed" dialog box when the service is installed successfully. This is sometimes not desired during installations, and in these cases you can use the /silent command-line switch to suppress the dialog box:

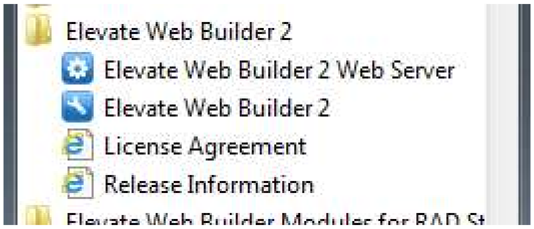
```
ewbsrvr.exe /install /silent
```

Starting the Web Server

The main difference between starting the web server as a normal application and starting the web server as a Windows service is that the normal application can be started just like any other application in one of three ways: the Run dialog, the command-line, or a Start Menu program link or desktop link. The service, however, must be started via the Services dialog or by using the NET START command-line command.

Starting the Web Server as a Normal Application

You can start the web server as a normal application by clicking on the link for the "Elevate Web Builder 2 Web Server".



Starting the Web Server as a Service

To start the web server as a Windows service, you can use the NET START command from the command-line:

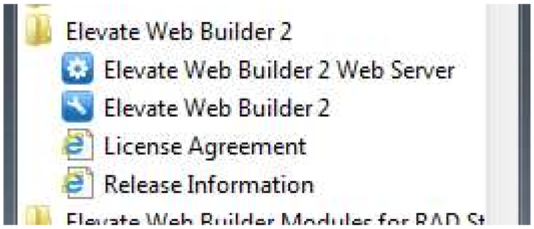
```
net start ewbsrvr
```

Note
In order to start the web server as a Windows service the server must have already been installed as a service using the /install command-line switch (see above).

7.2 Configuring the Web Server

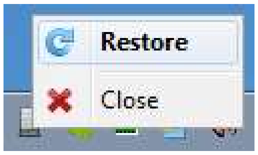
You can configure the web server by completing the following steps.

1. Start the web server (ewbsrvr.exe) as an application by clicking on the link for the "Elevate Web Builder 2 Web Server".

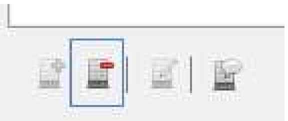


2. Access the web server configuration options:

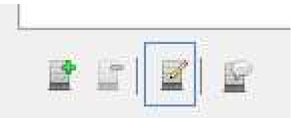
a. In the system tray, right-click on the web server icon to bring up the server menu, and click on the Restore option on the server menu.



b. Using the main toolbar, click on the **Stop Server** button.

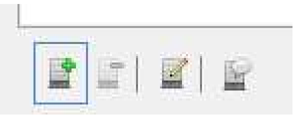


c. Using the main toolbar, click on the **Configure Server** button to open the Server Configuration Dialog.



d. Use the information below under the **Server Configuration Dialog** heading to configure the web server according to your needs. When done configuring the server, click on the **OK** button to save the changes.

e. Using the main toolbar, click on the **Start Server** button.

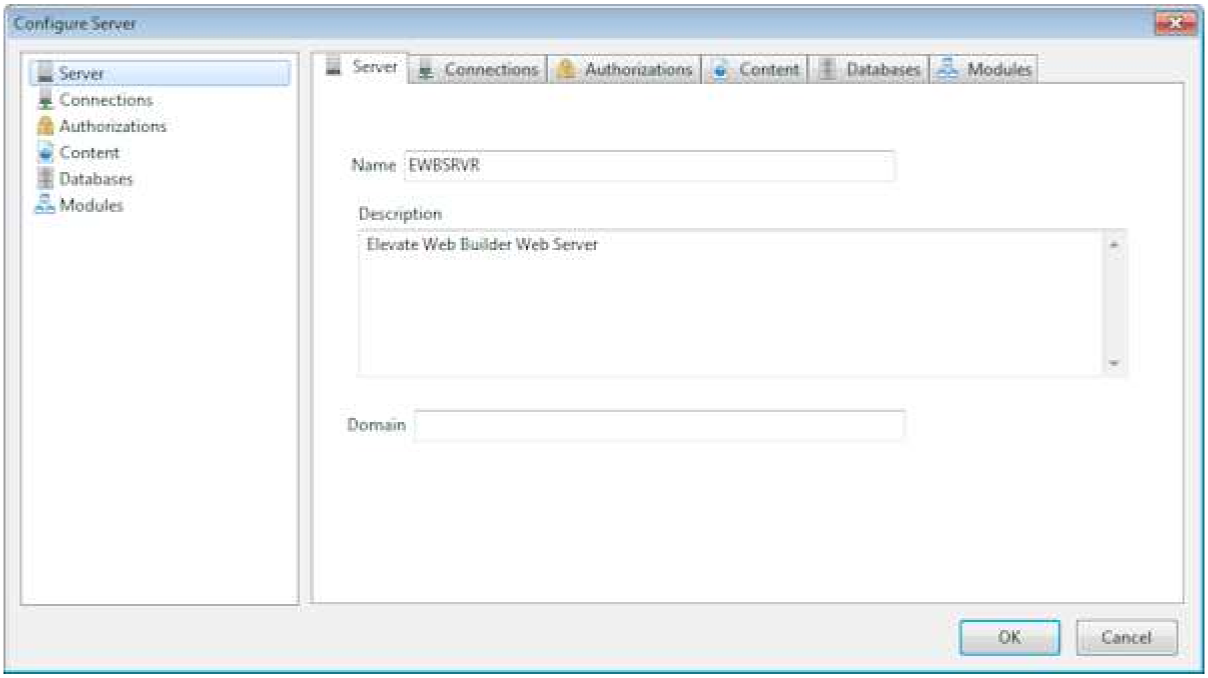


f. Click on the close button in the upper-right-hand corner of the web server window to close the server window.

Server Configuration Dialog

The Server Configuration dialog allows you to configure the following aspects of the web server:

- The server name, description, and hosted domain
- The IP address, port, and timeout settings for connections
- The authorization information for incoming connections
- The content folder, default document, cross-origin resource sharing, and resource names
- The databases and datasets defined for the web server
- The modules added to the web server



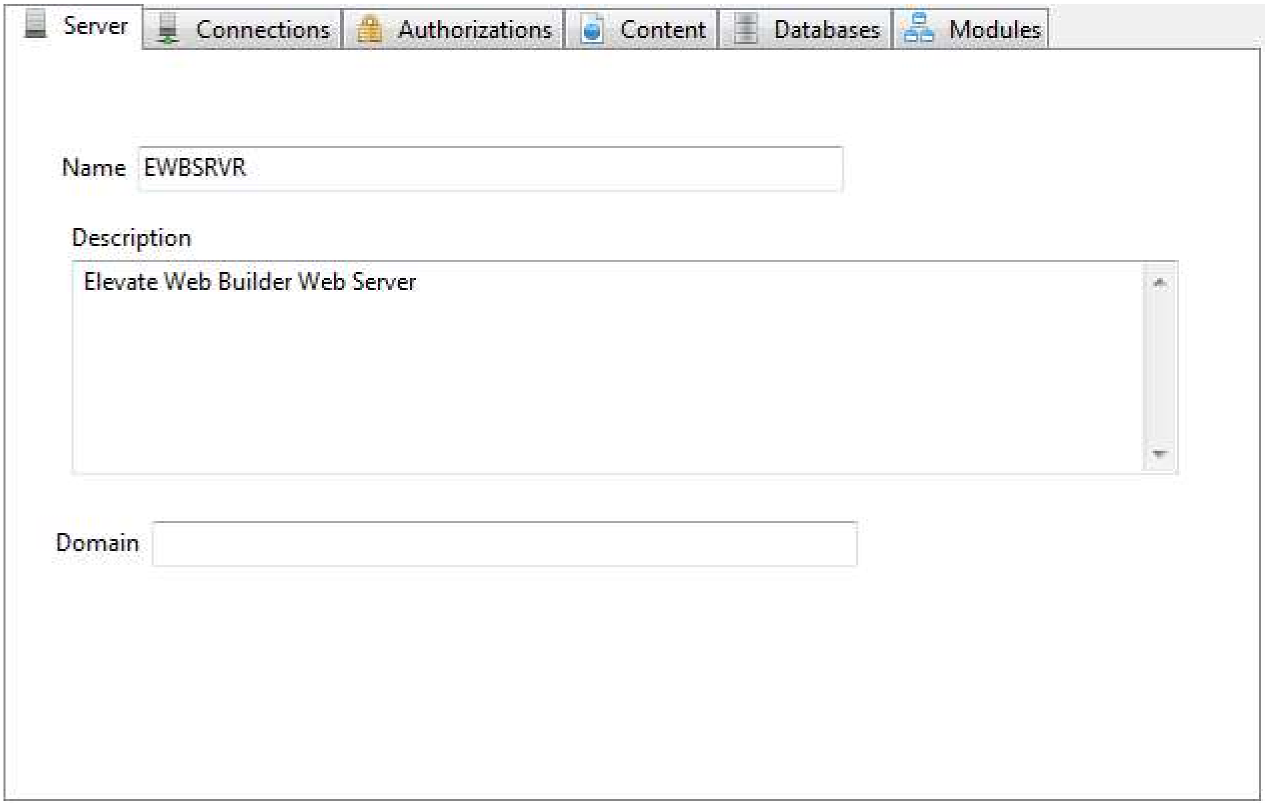
Name EWBSRVR

Description
Elevate Web Builder Web Server

Domain

Server

The Server page provides options for modifying the general web server settings.



Name EWBSRVR

Description

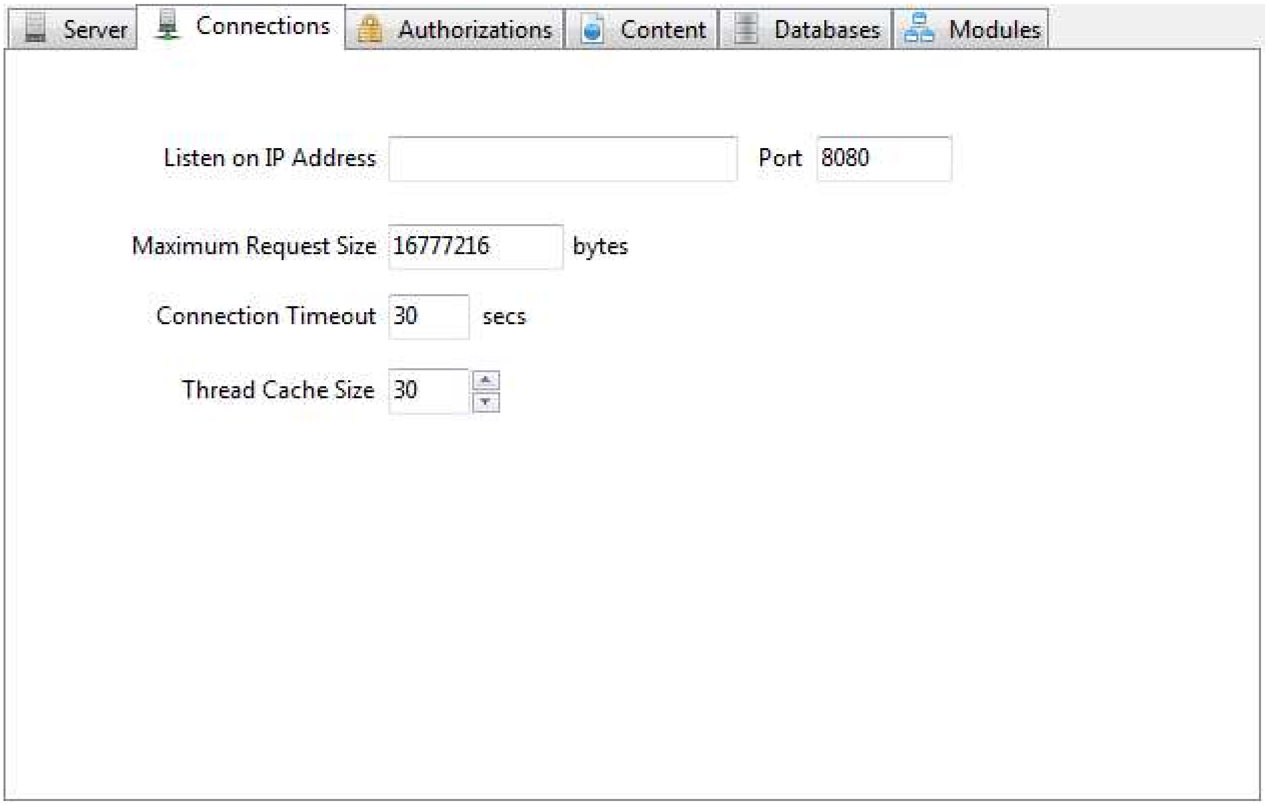
Elevate Web Builder Web Server

Domain

Option	Description
Name	Identifies the web server. This value is not used for named server instances (see below Multiple Server Instances for more information on named server instances). The default value is 'ewbsrvr'
Description	Used in conjunction with the name to give more information about the web server to clients once they have connected to the web server. The default value is 'Elevate Web Builder 2 Web Server'.
Domain	Identifies the domain that is being hosted by the web server. The default value is '', meaning that any domain can be hosted by the web server. If a value is provided, then it will be used to limit connections to the web server to those that pass the same value (case-insensitive) in the Host header along with each HTTP request.

Connections

The Connections page provides options for modifying the connection settings for the web server.



Listen on IP Address Port 8080

Maximum Request Size 16777216 bytes

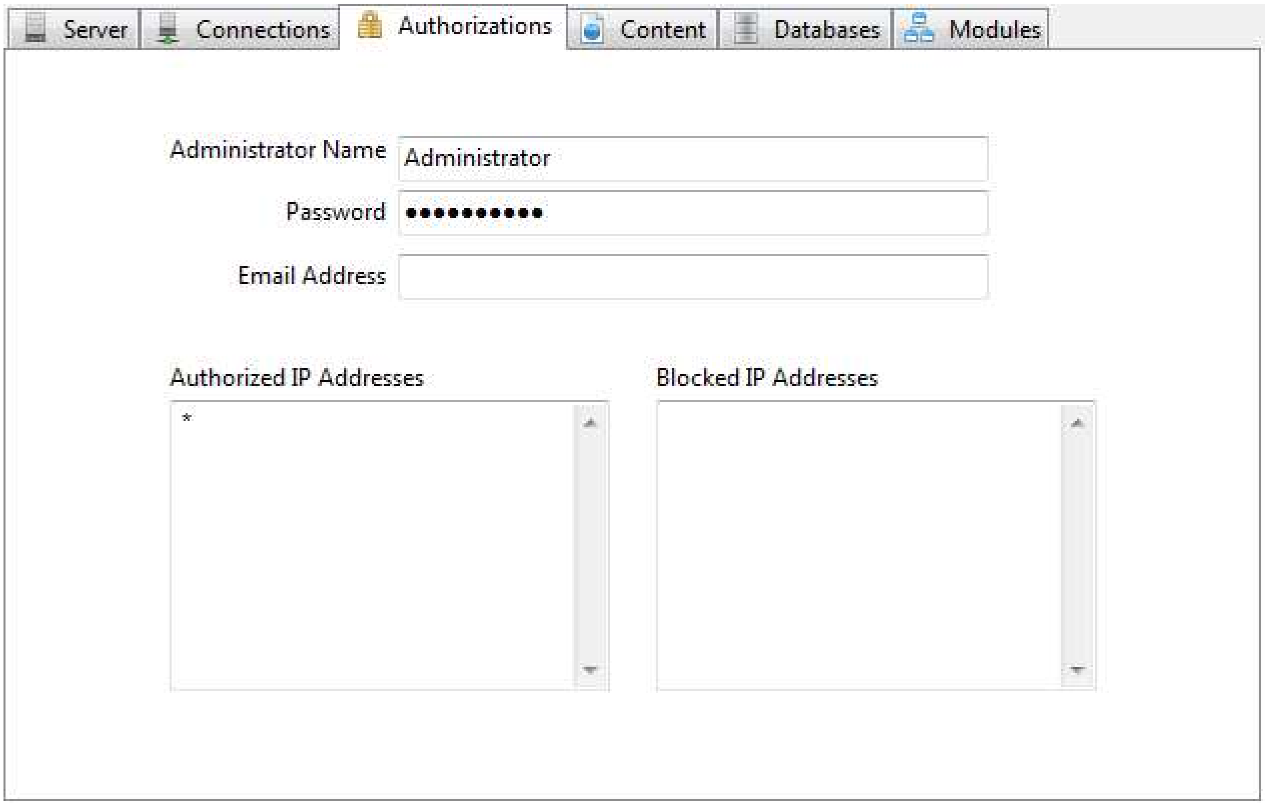
Connection Timeout 30 secs

Thread Cache Size 30

Option	Description
IP Address	Specifies the IP address that the web server should bind to when listening for incoming connections from clients. The default value is blank (""), which specifies that the web server should bind to all available IP addresses.
Port	Specifies the port that the web server should bind to when listening for incoming connections from clients. The default value is 80.
Maximum Request Size	Specifies the maximum size of any incoming request from a client. Clients will often send content as part of HTTP POST requests, and this setting acts as a governor to prevent a client from intentionally or unintentionally crashing the web server by exhausting all available memory. The default value is 16,777,216 bytes, or 16MB.
Connection Timeout	Specifies how long the web server should wait for a request from a connection before it terminates the connection. This is done to keep the number of concurrent connections to a minimum, while still allowing for multiple web browser requests on the same connection. The default value is 30 seconds.
Thread Cache Size	Specifies the number of threads that the web server should actively cache for connections. When a thread is terminated in the server it will be added to this thread cache until the number of threads cached reaches this value. This allows the web server to re-use the threads from the cache instead of having to constantly create/destroy the threads as needed, which can improve the performance of the web server if there are many connections and disconnections occurring. The default value is 10.

Authorizations

The Authorizations page provides options for modifying the authorization information for the web server.



Administrator Name: Administrator

Password: ●●●●●●●●●●●

Email Address:

Authorized IP Addresses

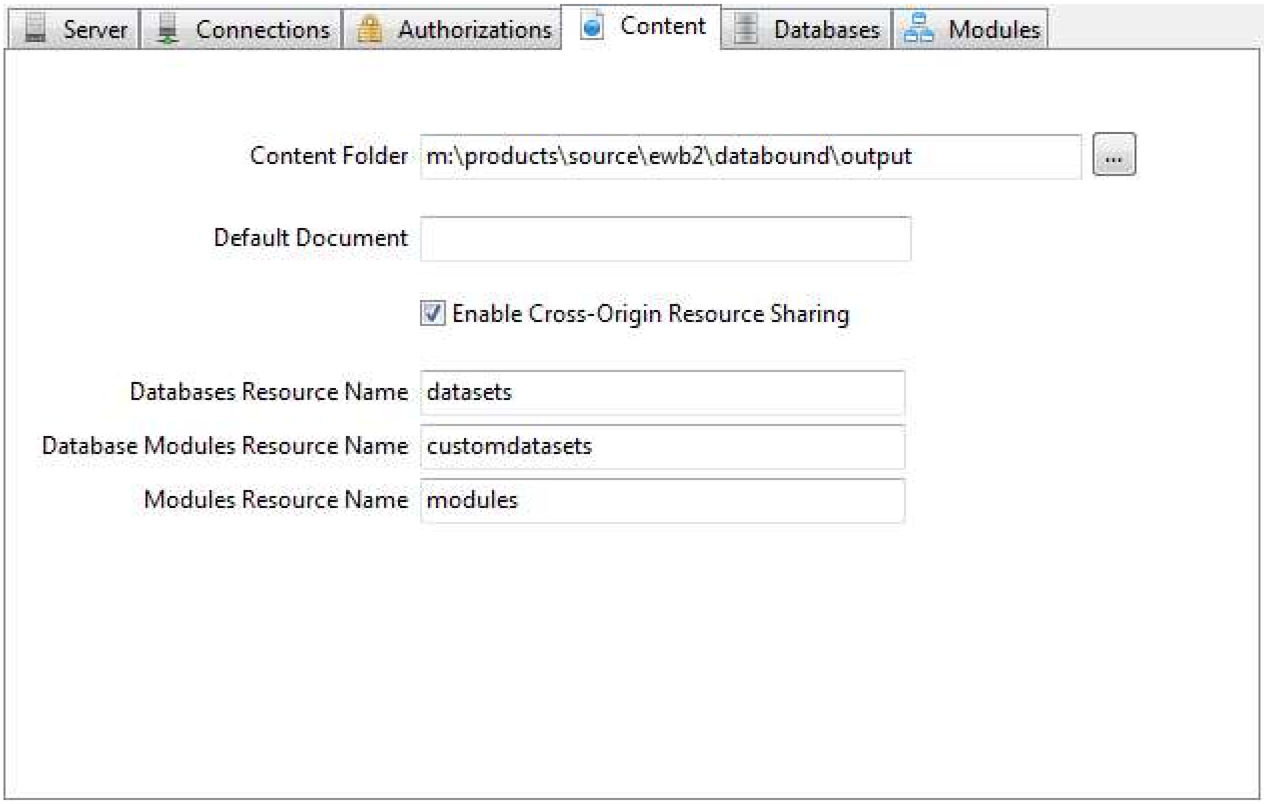
```
*
```

Blocked IP Addresses

Option	Description
Administrator Name	Specifies the administrator's name. The default value is 'Administrator'. **Note** The administrator name and password are not used currently, but will be in the future for remote administration of the web server.
Administrator Password	Specifies the administrator's password. The default value is 'EWBDefault'. **Note** The administrator name and password are not used currently, but will be in the future for remote administration of the web server.
Administrator Email	Specifies the administrator's email address, which is used in web server response headers to indicate the point of contact for the organization in case of errors, etc. The default value is ''.
Authorized IP Addresses	Specifies which IP addresses are authorized to access the web server. This is commonly referred to as a "white list". There is no limit to the number of addresses that can be specified, and the IP address entries may contain the asterisk (*) wildcard character to represent any portion of an address. **Note** Due to the way that .ini file entries must be specified, multiple addresses must be separated with the literal value "<#CR#><#LF#>" (without the quotes) instead of actual line feeds.
Blocked IP Addresses	Specifies which IP addresses are not allowed to access the web server. This is commonly referred to as a "black list". There is no limit to the number of addresses that can be specified, and the IP address entries may contain the asterisk (*) wildcard character to represent any portion of an address. **Note** Due to the way that .ini file entries must be specified, multiple addresses must be separated with the literal value "<#CR#><#LF#>" (without the quotes) instead of actual line feeds.

Content

The Content page provides options for modifying the content settings for the web server.



Content Folder m:\products\source\ewb2\databound\output

Default Document

☑ Enable Cross-Origin Resource Sharing

Databases Resource Name datasets

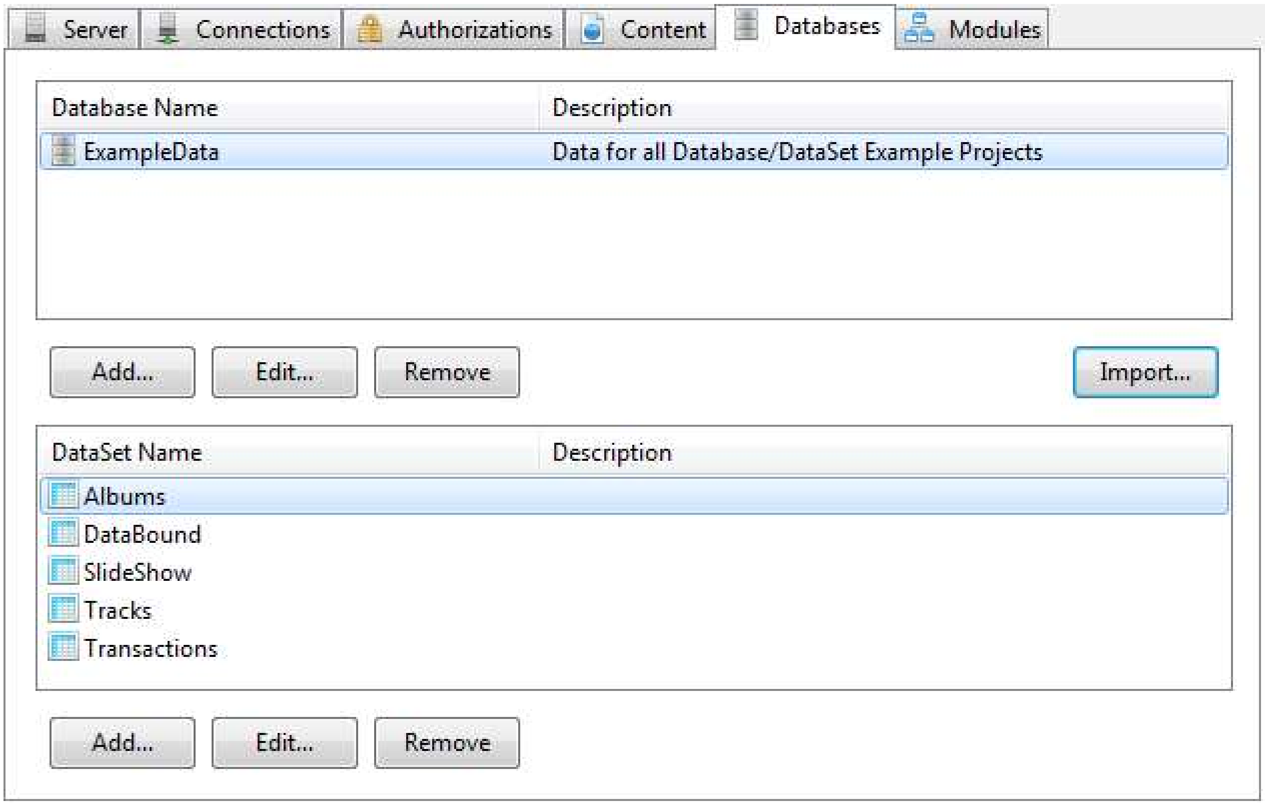
Database Modules Resource Name customdatasets

Modules Resource Name modules

Option	Description
Content Folder	Specifies the path that the web server should use for all static content, including Elevate Web Builder applications (*.js, *.html) and any other external files such as images. The default value is ''.
Default Document	Specifies the default document file name to use if a URL requested by a client does not include a file name. For example, if the client makes a request to the URL "http://www.mydomain.com", then this value will be appended to the URL (prefixed with a slash) and the combined URL will be used instead. The default value is ''.
Enable Cross-Origin Resource Sharing	Specifies that the web server will allow and handle cross-origin resource sharing. This feature allows the web server to serve static and database content in response to requests from origins (domain name and port number) that are different than that of the web server. Normally, web browsers don't permit such cross-origin requests unless the web server specifically allows them. The default value is False. For a good discussion of Cross-Origin Resource Sharing, please visit the following link: HTTP access control (CORS) **Note** The Elevate Web Builder Web Server supports both simple and preflighted requests, but does not support requests with credentials at this time.
Databases Resource Name	Specifies the resource name to use for the automatic database handling built into the web server. The default value is 'databases'. Please see the Web Server Request Handling topic for more information on how this resource name is used in database requests.
Database Modules Resource Name	Specifies the resource name to use for any database modules added to the web server (see next). The default value is 'databasemodules'. Please see the Web Server Request Handling topic for more information on how this resource name is used in database requests.
Modules Resource Name	Specifies the resource name to use for any web server modules added to the web server. The default value is 'modules'. Please see the Web Server Request Handling topic for more information on how this resource name is used in module requests.

Databases

The Databases page provides options for modifying the defined databases and datasets used by the web server.



Database Name	Description
ExampleData	Data for all Database/DataSet Example Projects

DataSet Name	Description
Albums	
DataBound	
SlideShow	
Tracks	
Transactions	

Adding a New Database

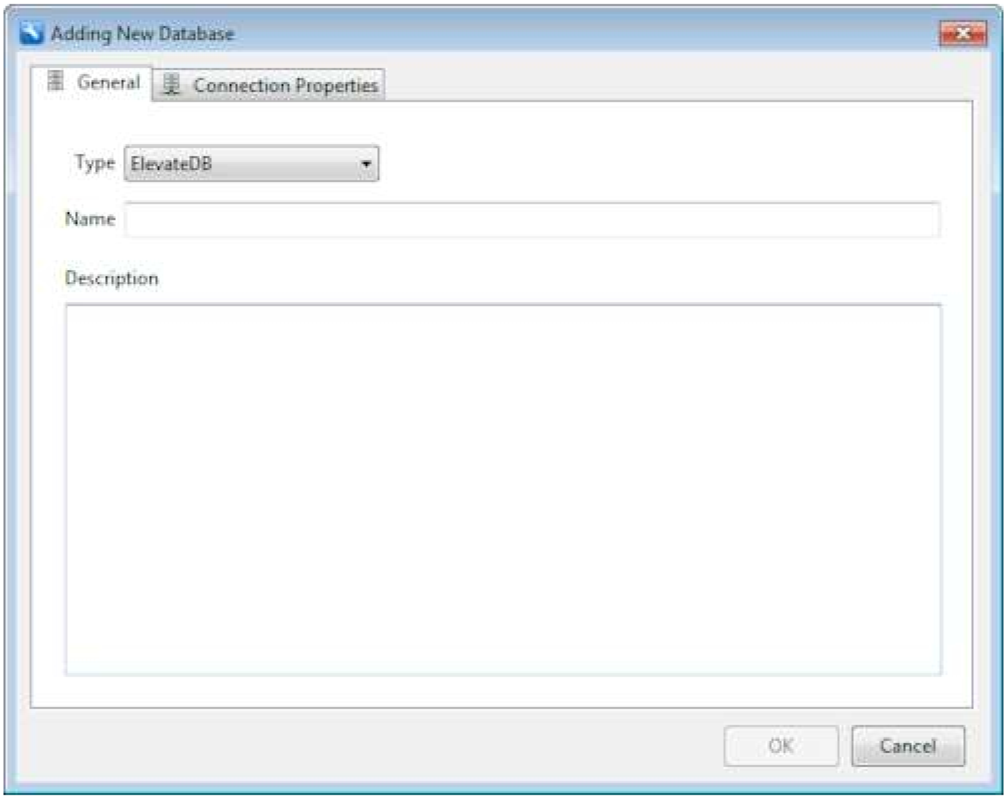
Use the following steps to add a new database:

- Click on the Add button under the list of databases.
- The database editor dialog will appear. Please refer to the next section for information on defining the database.

Defining a Database

The database editor dialog consists of 2 pages:

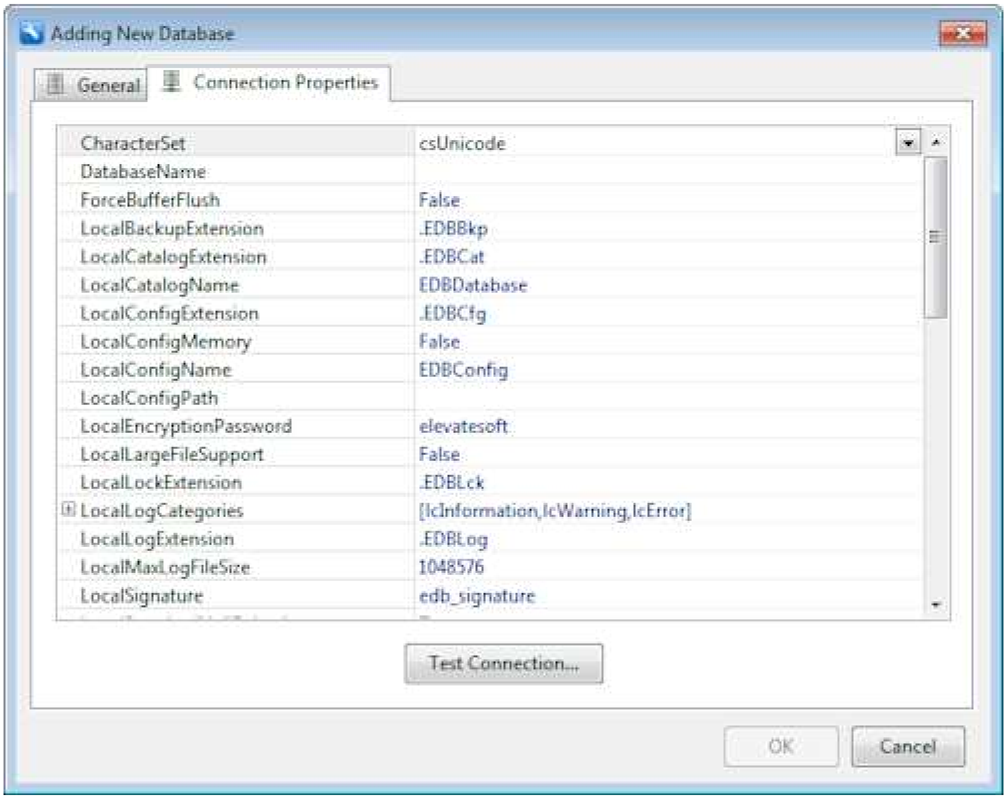
- **General** - the database engine/server type, the name of the database, and the description.



Currently, the following database engines are supported:

ElevateDB
DBISAM
ADO (includes OLEDB/ODBC)

- **Connection Properties** - the name/location of the database and other configuration properties essential to establishing a proper connection to the desired database. The options on this page are specific to the database engine selected on the first page.



CharacterSet	csUnicode
DatabaseName	
ForceBufferFlush	False
LocalBackupExtension	.EDBBkp
LocalCatalogExtension	.EDBCat
LocalCatalogName	EDBDatabase
LocalConfigExtension	.EDBCfg
LocalConfigMemory	False
LocalConfigName	EDBConfig
LocalConfigPath	
LocalEncryptionPassword	elevatesoft
LocalLargeFileSupport	False
LocalLockExtension	.EDBLck
⊞ LocalLogCategories	[lcInformation,lcWarning,lcError]
LocalLogExtension	.EDBLog
LocalMaxLogFileSize	1048576
LocalSignature	edb_signature

Once the connection properties are set, you can use the Test Connection button to verify that everything is set properly. Please see your database engine manual/documentation for more information on the proper value for each property setting.

- Once you have properly set the connection properties and successfully tested the connection to the database, click on the OK button to close the database dialog and save the database.

Editing an Existing Database

To edit an existing database, simply double-click on the desired database in the list of databases. The database editor dialog will then appear, and you can use it to modify the database accordingly.

Removing a Database

Use the following steps to remove a database:

- Click on the name of the database that you wish to remove.

- Click on the Remove button under the list of databases.

- A confirmation dialog will be displayed, asking you to confirm the removal of the database. Click on the Yes button to continue, or the No button to cancel the removal.

Importing Databases from the IDE

In order to import databases from the Elevate Web Builder IDE, both the IDE and the web server must be running on the same machine. The import process currently only imports the database definitions directly from the IDE's .ini file.

Warning
Importing the databases from the IDE will cause all existing databases defined for the web server to be replaced with those from the IDE. Please make sure that this is the desired outcome before proceeding.

To begin the import process, simply click on the Import button under the list of databases.

Adding a New DataSet

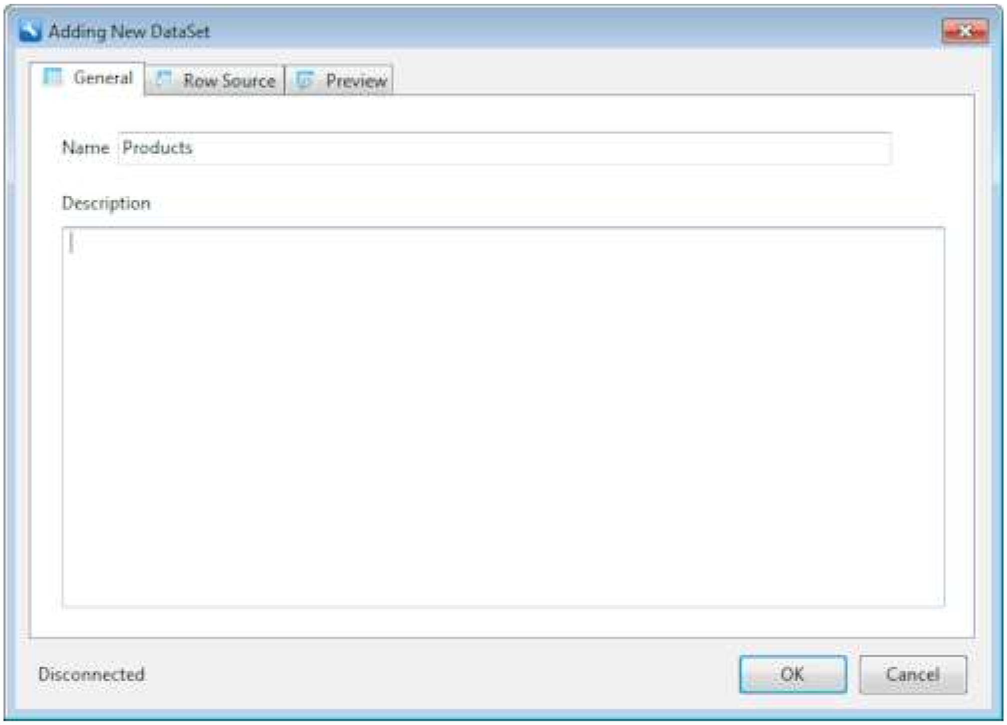
Use the following steps to add a new dataset:

- Be sure that you have selected an existing database by clicking on the desired existing database.
- Click on the Add button under the list of datasets for the currently-selected database.
- The dataset editor dialog will appear. Please refer to the next section for information on defining the dataset.

Defining a DataSet

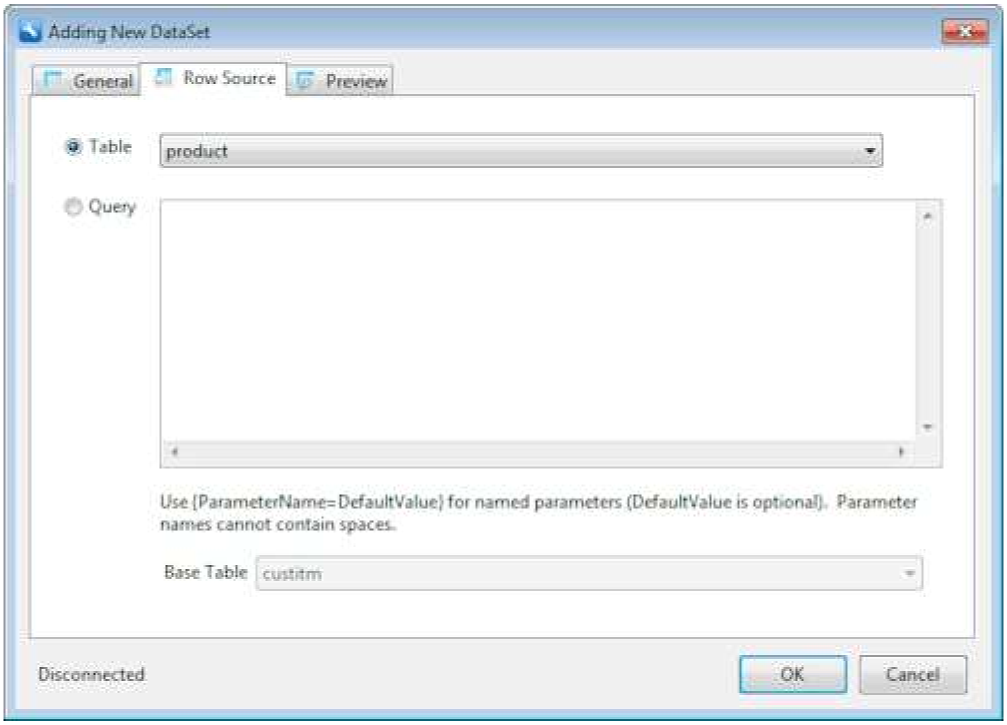
The dataset editor dialog consists of 3 pages:

- **General** - the name of the dataset and the description.



Name: Products

- **Row Source** - the actual source of the dataset rows can be an actual table name from the selected database, or it can be an SQL SELECT statement.



● Table: product

○ Query

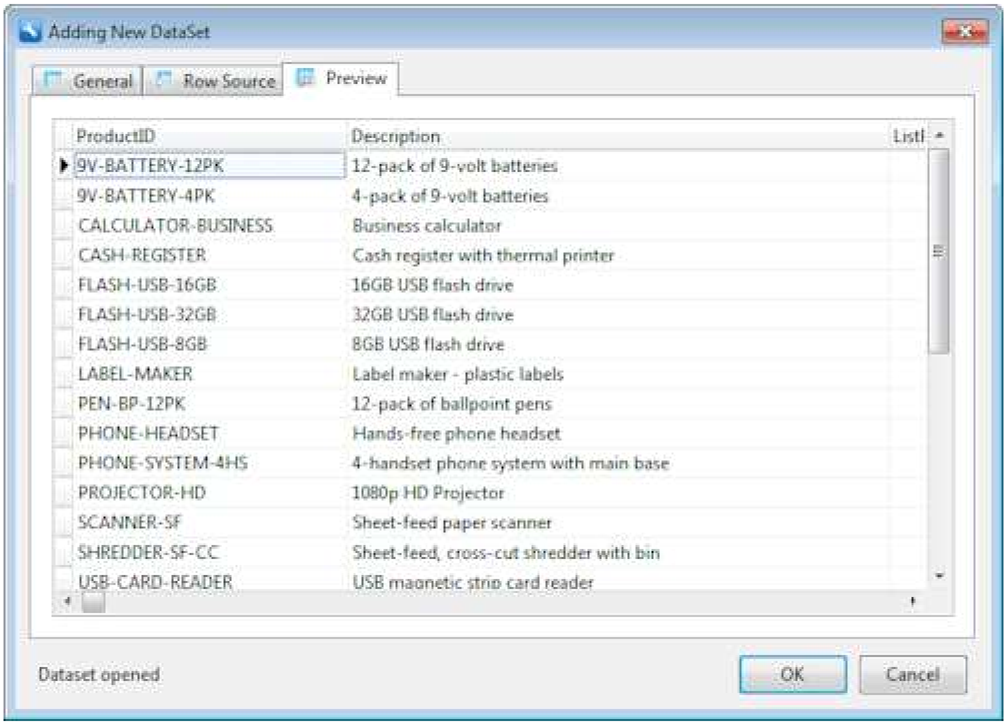
Use {ParameterName=DefaultValue} for named parameters (DefaultValue is optional). Parameter names cannot contain spaces.

Base Table: custitm

Disconnected

Elevate Web Builder uses a special parameter naming syntax for queries, and does **not** use the native parameter functionality in the target database engine. This is done because some database engines do not support named parameters, or do not support parameter type discovery or enumeration. When the dataset rows are requested from the internal web server embedded in the IDE, it automatically populates the named parameters in the query by using the URL "name=value" parameters passed with the dataset rows request. These parameters can be specified in the application via the TDataSet Params property.

- **Preview** - use the preview page to make sure that the dataset is returning the correct rows. Any default values for parameters defined on the Row Source page are applied for the preview, so if you have not defined any default parameter values you may see zero rows displayed.



ProductID	Description	Listl
9V-BATTERY-12PK	12-pack of 9-volt batteries	
9V-BATTERY-4PK	4-pack of 9-volt batteries	
CALCULATOR-BUSINESS	Business calculator	
CASH-REGISTER	Cash register with thermal printer	
FLASH-USB-16GB	16GB USB flash drive	
FLASH-USB-32GB	32GB USB flash drive	
FLASH-USB-8GB	8GB USB flash drive	
LABEL-MAKER	Label maker - plastic labels	
PEN-BP-12PK	12-pack of ballpoint pens	
PHONE-HEADSET	Hands-free phone headset	
PHONE-SYSTEM-4HS	4-handset phone system with main base	
PROJECTOR-HD	1080p HD Projector	
SCANNER-SF	Sheet-feed paper scanner	
SHREDDER-SF-CC	Sheet-feed, cross-cut shredder with bin	
USB-CARD-READER	USB magnetic strip card reader	

Editing an Existing DataSet

To edit an existing dataset, simply double-click on the desired dataset in the list of datasets. The dataset editor dialog will then appear, and you can use it to modify the dataset accordingly.

Removing a DataSet

Use the following steps to remove a dataset:

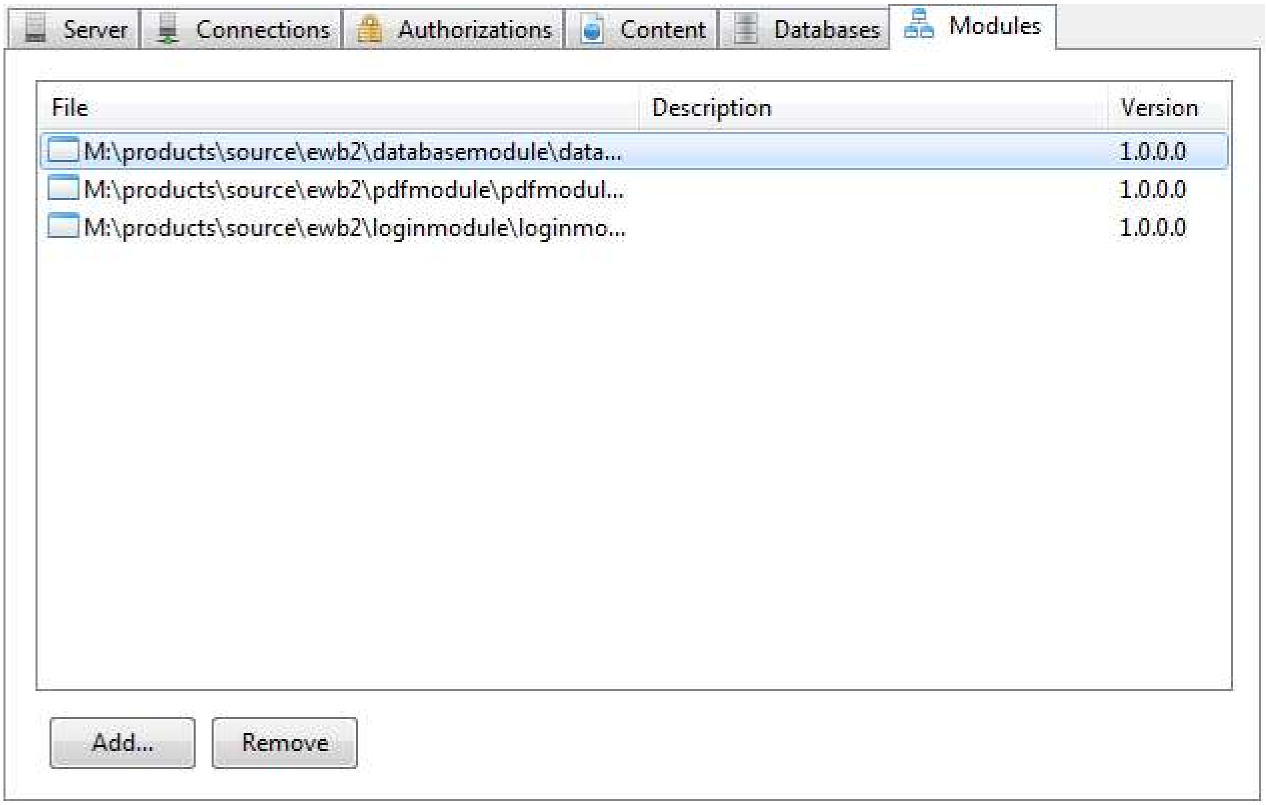
- Click on the name of the dataset that you wish to remove.

- Click on the Remove button under the list of datasets for the currently-selected database.

- A confirmation dialog will be displayed, asking you to confirm the removal of the dataset. Click on the Yes button to continue, or the No button to cancel the removal.

Modules

The Modules page provides options for adding and removing modules (*.dll) that were created using Embarcadero RAD Studio and Delphi and an Elevate Web Builder Module template project from the repository in the RAD Studio IDE. Adding modules to the web server allows the modules to be used to respond to requests and provide content to the Elevate Web Builder application running in the web browser.

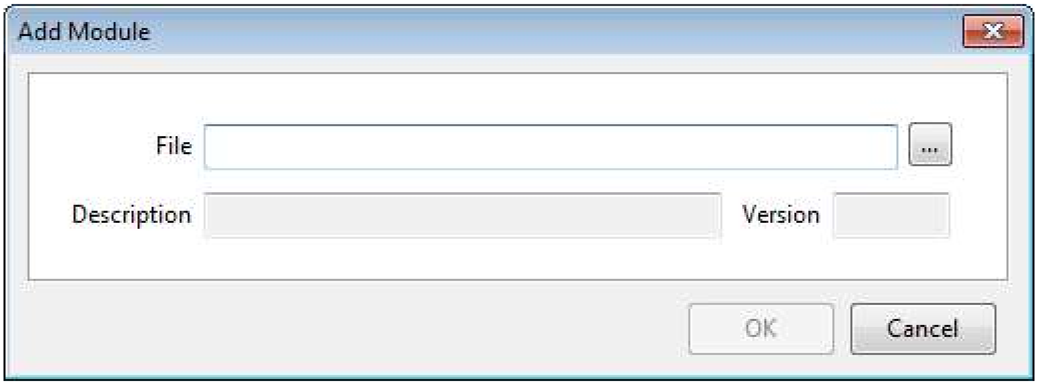


File	Description	Version
M:\products\source\ewb2\databasemodule\data...		1.0.0.0
M:\products\source\ewb2\pdfmodule\pdfmodul...		1.0.0.0
M:\products\source\ewb2\loginmodule\loginmo...		1.0.0.0

Adding a Module

In order to add a module, complete the following steps:

- Click on the Add button
- The Add Module dialog will appear.



In the dialog, specify the file name of the module (.dll) that you wish to add to the web server. You can type in the file name directly, or use the browse button (...) to select the module using a common Windows file dialog. If you use the browse button, the module description and version will be populated from the module after the file is selected. The description and version are read directly from the .dll's version information.

- Click on the OK button. If the specified file is a valid Elevate Web Builder module, then the module will be added to the web server. If the specified file is not a valid module file, then an error message will be displayed indicating any issues with the module file.

Removing a Module

In order to remove a module, complete the following steps:

- Select an existing module from the list of modules.

- Click on the Remove button.

Note
If you remove a module that is used by Elevate Web Builder applications, then you will experience errors in these applications when they try to execute requests that reference these modules in the URL for the request.

Please see the Creating Web Server Modules topic for more information how the modules work.

Configuration Reference

The web server stores its configuration information in an .ini file that is, by default, located in the following directory under Windows XP/2003 Server:

```
C:\Documents and Settings\All Users\Application Data\Elevate Software\Elevate
      Web Builder 2 Web Server
```

in the following directory under Windows Vista or higher (including Windows 7 and Server 2008):

```
C:\ProgramData\Elevate Software\Elevate Web Builder 2 Web Server
```

The name of the .ini configuration file is determined by the name of the application. For example, for the ewbsrvr.exe application, the name of the .ini file would be ewbsrvr.ini.

Note
If the web server finds an .ini with the proper name in the same directory as the server .exe, it will use it instead of the .ini file in the common application data directory for Windows.

All of the configuration entries in the web server .ini configuration files are stored under a section called "Server".

Note
Please see the Multiple Server Instances topic for how multiple server instances can change this naming slightly.

Each of the individual configuration entries in this section are as follows:

Configuration Entry	Description
No User Interface	Specifies that the server will run without a user interface. This is useful in situations where you don't want the server to display an interface or an icon in the system tray, such as when running the server as a Windows service. The default value is 0 (False). Setting this entry to 1 (True) will turn off the server UI.
Server Name	Identifies the web server. This configuration item is not used for named server instances (see below Multiple Server Instances for more information on named server instances). The default value is 'ewbsrvr'.
Server Description	Used in conjunction with the "Server Name" configuration entry to give more information about the web server to external clients once they have connected to the web server. The default value is 'Elevate Web Builder Web Server'.
Domain	Identifies the domain that is being hosted by the web server. The default value is '', meaning that any domain can be hosted by the web server. If a value is provided, then it will be used to limit connections to the web server to those that pass the same value (case-insensitive) in the Host header along with each HTTP request.
IP Address	Specifies the IP address that the web server should bind to when listening for incoming connections from web browsers. The default value is blank (""), which specifies that the web server should bind to all available IP addresses.
Port	Specifies the port that the web server should bind to when listening for incoming connections from web browsers. The default value is 80.
Max Request Size	Specifies the maximum size of any incoming request from a client. Clients will often send content as part of HTTP POST requests, and this setting acts as a governor to prevent a client from intentionally or unintentionally crashing the web server by exhausting all available memory. The default value is 16,777,216 bytes, or 16MB.
Timeout	Specifies how long the web server should wait for a request from a connection before it terminates the connection. This is done to keep the number of concurrent connections to a minimum, while still allowing for multiple web browser requests on the same connection. The default value is 30 seconds.
Thread Cache Size	Specifies the number of threads that the web server should actively cache for connections. When a thread is terminated in

	the server it will be added to this thread cache until the number of threads cached reaches this value. This allows the web server to re-use the threads from the cache instead of having to constantly create/destroy the threads as needed, which can improve the performance of the web server if there are many connections and disconnections occurring. The default value is 10.
Admin Name	Specifies the administrator's name. The default value is 'Administrator'. **Note** The administrator name and password are not used currently, but will be in the future for remote administration of the web server.
Admin Password	Specifies the administrator's password. The default value is 'EWBDefault'. **Note** The administrator name and password are not used currently, but will be in the future for remote administration of the web server.
Admin Email	Specifies the administrator's email address, which is used in web server response headers to indicate the point of contact for the organization in case of errors, etc. The default value is ''.
Authorized Addresses	Specifies which IP addresses are authorized to access the web server. This is commonly referred to as a "white list". There is no limit to the number of addresses that can be specified, and the IP address entries may contain the asterisk (*) wildcard character to represent any portion of an address. **Note** Due to the way that .ini file entries must be specified, multiple addresses must be separated with the literal value "<#CR#><#LF#>" (without the quotes) instead of actual line feeds.
Blocked Addresses	Specifies which IP addresses are not allowed to access the web server. This is commonly referred to as a "black list". There is no limit to the number of addresses that can be specified, and the IP address entries may contain the asterisk (*) wildcard character to represent any portion of an address.

	Note Due to the way that .ini file entries must be specified, multiple addresses must be separated with the literal value "<#CR#><#LF#>" (without the quotes) instead of actual line feeds.
Content Folder	Specifies the path that the web server should use for all static content, including Elevate Web Builder applications (*.js, *.html) and external files like images. The default value is ''.
Default Document	Specifies the default document file name to use if a URL requested by a client does not include a file name. For example, if the client makes a request to the URL "http://www.mydomain.com", then this value will be appended to the URL (prefixed with a slash) and the combined URL will be used instead. The default value is ''.
Enable Cross Origin Resources	Specifies that the web server will allow and handle cross-origin resource sharing. This feature allows the web server to serve static and database content in response to requests from origins (domain name and port number) that are different than that of the static and database content. Normally, web browsers don't permit such cross-origin requests unless the web server specifically allows them. The default value is False.
Databases Resource Name	Specifies the resource name to use for the automatic database handling built into the web server. The default value is 'databases'. Please see the Web Server Request Handling topic for more information on how this resource name is used in database requests.
Database Modules Resource Name	Specifies the resource name to use for any database modules added to the web server (see next). The default value is 'databasemodules'. Please see the Web Server Request Handling topic for more information on how this resource name is used in database requests.
Modules Resource Name	Specifies the resource name to use for any web server modules added to the web server. The default value is 'modules'. Please see the Web Server Request Handling topic for more information on how this resource name is used in module requests.

7.3 Multiple Web Server Instances

Multiple instances of the web server can be run on the same physical machine through named server instances. Named server instances are simply instances of the web server that were executed using two special command-line switches:

```
ewbsrvr.exe /name=<Server Instance Name> /desc=<Server Instance Description>
```

Named server instances use the passed name and description to provide the name of the web server instance, as well as the description. The name parameter is also used to determine which section of the ewbsrvr.ini file is used for configuration purposes. Instead of just the normal "Server" section being used in the ewbsrvr.ini file, the section is named using the provided server name. For example, if the named server instance is called "MyServer", then the section in the ewbsrvr.ini file where the configuration is stored will be the following:

```
[Server_MyServer]
```

> **Note**
> This also applies to the defined datasets and modules in the ewbsrvr.ini file. Each named server instance will have its own datasets and modules. Please see the Configuring the Web Server topic for more information on configuring the web server.

The description parameter, if also specified, is immediately written to the named server instance section of the ewbsrvr.ini file. All other configuration options described above in the Configuration Reference must be modified using the Server Configuration Dialog in the web server. You can run the web server as a normal application in order to modify the configuration of a named server instance. For example, to modify the MyServer configuration you would use the following from the command-line:

```
ewbsrvr.exe /name=MyServer"
```

In order to use a named server instance as a Windows service, the /name parameter must be specified during the installation of the service. For example, if the named server instance is called "MyServer", then the service installation would be accomplished using the following from the command-line:

```
ewbsrvr.exe /install /name=MyServer /desc="My Server"
```

When you want to start the named server instance as a service, you would simply just use the following from the command-line:

```
net start MyServer
```

The following example shows how you would install two web server named server instances as Windows services, and then start them:

```
ewbsrvr.exe /install /name=MyFirstServer /desc="My First Server"

ewbsrvr.exe /install /name=MySecondServer /desc="My Second Server"

net start MyFirstServer

net start MySecondServer
```

Warning

You will need to verify that the port being used by each named server instance is unique, or one or more named server instances will not start due to a port conflict. As mentioned above, you can use the web server run as a normal application to modify the configuration of any named server instance.

7.4 Web Server Request Handling

As discussed in the Server Request Architecture topic, HTTP requests are usually HEAD, GET, or POST requests, but can also be PUT or DELETE requests with defined REST interfaces. In addition, there are certain types of HTTP requests that are automatically handled by the web server.

Note
All URL comparisons performed in the web server are case-insensitive.

HEAD and GET Requests

With any HTTP HEAD or GET request, the web server does the following:

- It first checks the URL to see if the request is for static content such as an HTML, JavaScript, image, etc. file. The content folder specified in the web server configuration is used as the root folder for this check. If a file is found in the location specified by the URL, relative to the content folder, and the request is an HTTP GET request, then the file is sent to the client via the HTTP response headers and content. If the request is an HTTP HEAD request, then only the HTTP response headers are sent and not the file content.

- If the URL is not that of static content, it will then be checked to see if it is an Elevate Web Builder database request. A database request is any request that uses the following URL structure:

```
http://<Domain Name>/<Databases Resource Name>?method=<Method
      Name>&database=<Database Name>[&dataset=<DataSet Name>] or
https://<Domain Name>/<Databases Resource Name>?method=<Method
      Name>&database=<Database Name>[&dataset=<DataSet Name>]
```

  If the URL matches this pattern, then the web server will automatically handle such a request and return a proper response to the client.

  Note
  Database requests are never HTTP HEAD requests, only GET or POST requests.

- If the URL is not that of static content or a database request, then it is then checked to see if it is an Elevate Web Builder server module request. A module request is any request that uses one of the following URL structures:

  Normal Module

```
http://<Domain Name>/<Modules Resource Name>/<Module Name> or
https://<Domain Name>/<Modules Resource Name>/<Module Name>
```

Database Module

```
http://<Domain Name>/<Database Modules Resource Name>/<Database Module Name>
      or
https://<Domain Name>/<Database Modules Resource Name>/<Database Module Name>
```

> **Note**
> The <Module Resource Name> and <Database Module Resource Name> components of the URL are the default resource names for modules and database modules defined in the Elevate Web Builder Web Server, but can be changed in the web server configuration. If you've changed the default modules resource name of 'modules', then please replace any subsequent references to the default 'modules' resource name in the following examples with the resource name that you're using instead. The same holds true for the default database modules resource name of 'databasemodules'. Please see the Configuring the Web Server topic for more information.

If the URL matches this pattern, then the web server will automatically instantiate a module for use with the request and pass the request information to the module. Please see your product-specific module manual for information on how to handle such a request in a module and return a response.

POST Requests

With any HTTP POST request, the web server does the following:

- The URL is checked to see if it is an Elevate Web Builder database request. A database request is any request that uses the following URL structure:

```
http://<Domain Name>/<Databases Resource Name>?method=<Method
      Name>&database=<Database Name> or
https://<Domain Name>/<Databases Resource Name>?method=<Method
      Name>&database=<Database Name>
```

If the URL matches this pattern, then the web server will automatically handle such a request and return a proper response to the client.

> **Note**
> Database POST requests are always transactions, which is why the dataset name is not specified in the URL.

- If the URL is not that of a database request, then it is then checked to see if it is an Elevate Web Builder server module request. A module request is any request that uses one of the following URL structures:

  Normal Module

  ```
  http://<Domain Name>/<Modules Resource Name>/<Module Name> or
  https://<Domain Name>/<Modules Resource Name>/<Module Name>
  ```

  Database Module

  ```
  http://<Domain Name>/<Database Modules Resource Name>/<Database Module Name>
        or
  https://<Domain Name>/<Database Modules Resource Name>/<Database Module Name>
  ```

  If the URL matches this pattern, the web server will then automatically instantiate a module for use with the request and pass the request information to the module. Please see your product-specific module manual for information on how to handle such a request in a module and return a response.

PUT and DELETE Requests

With any HTTP PUT or DELETE request, the web server does the following:

- The URL is checked to see if it is an Elevate Web Builder server module request. A module request is any request that uses one of the following URL structures:

  Normal Module

  ```
  http://<Domain Name>/<Modules Resource Name>/<Module Name> or
  https://<Domain Name>/<Modules Resource Name>/<Module Name>
  ```

  Database Module

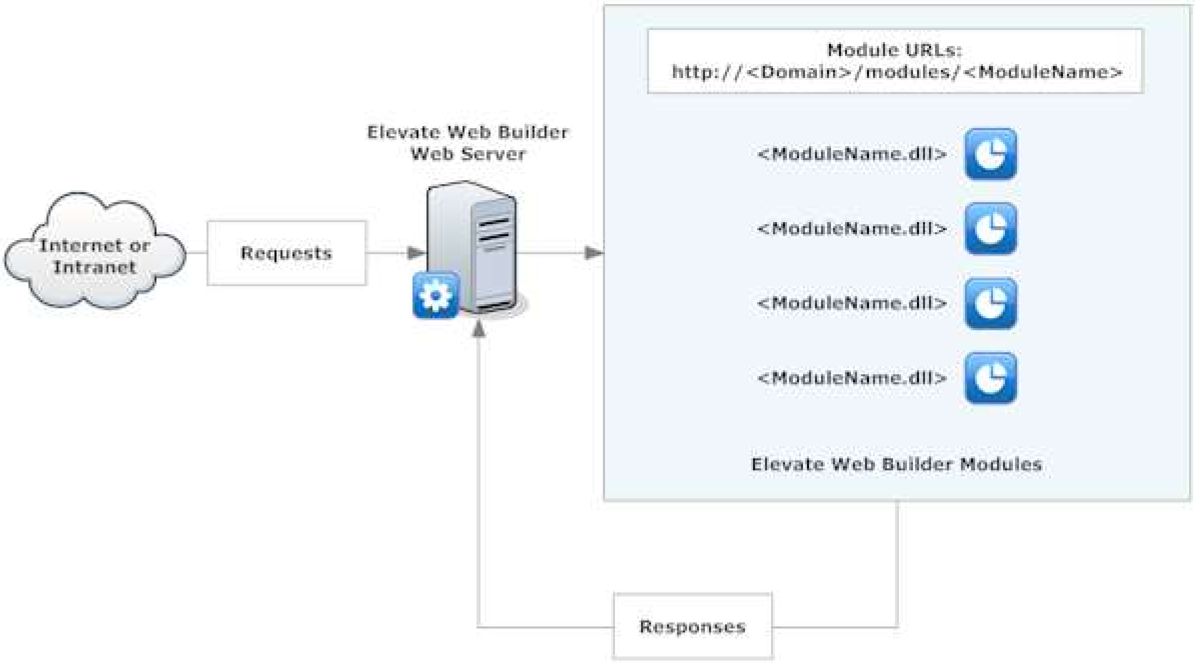
  ```
  http://<Domain Name>/<Database Modules Resource Name>/<Database Module Name>
        or
  https://<Domain Name>/<Database Modules Resource Name>/<Database Module Name>
  ```

  If the URL matches this pattern, the web server will then automatically instantiate a module for use with the request and pass the request information to the module. Please see your product-specific module manual for information on how to handle such a request in a module and return a response.

7.5 Creating Web Server Modules

The Elevate Web Builder Web Server allows modules created using Embarcadero RAD Studio and Delphi to be added to the web server and used to handle dynamic requests to the web server. This allows the developer to offload computationally-intensive or database-intensive work to the natively-compiled web server modules, and then use simple server requests in the front-end Elevate Web Builder applications to make requests to the modules and receive back responses along with the relevant data/content.

The web server module functionality has the following architecture:



Elevate Web Builder Modules Architecture

> **Note**
> While modules are actually DLLs with a .dll extension, module names are specified in URLs without any extension.

Creating Modules

For information on creating modules, please refer to the product-specific Elevate Web Builder 2 Modules Manual that accompanies the product installation. You can download and install the Elevate Web Builder 2 Modules product for your specific product using the following link:

Elevate Web Builder Downloads

Modules can only be created using Embarcadero RAD Studio and Delphi XE or higher. This is because the web server is using Unicode strings for all functionality and requires that the compiler used to create the modules use Unicode strings as the default string type. In addition, much of the support code that is used with the modules was developed using Delphi XE and contains references to code that is only present in Delphi XE or higher.

Adding Modules to the Web Server

For information on adding modules to the web server, please see the Configuring the Web Server topic.

Database Modules

Database modules are exactly the same as normal modules, but are referenced using a different resource name so that they can be kept logically separate from other modules. Please see the **Content** section of the Configuring the Web Server topic for information on configuring the database modules resource name in the web server. While structurally the same as a normal module, a database module always includes functionality for generating JSON data for loading datasets and consuming JSON data for database transactions. The majority of this code is already implemented for you via database and dataset adapter components that are made available with each Elevate Web Builder 2 Modules installation for Delphi XE or higher. These adapter components allow the developer to access/update a wider variety of data sources than what is currently possible with the built-in database functionality in the web server. However, the fact that a database module handles various database requests does not preclude it from also handling normal server requests. You can handle any normal requests before passing any database requests on to the database adapter for automatic handling.

> **Note**
> For more information on how to code a database module, please see the database module example installed in the \examples\databasemodule subdirectory along with the Elevate Web Builder 2 Modules download for the version of RAD Studio and Delphi that you are using.

This page intentionally left blank

Chapter 8
Language Reference

8.1 Introduction

Elevate Web Builder uses an Object Pascal dialect for its core language that is very close to the Object Pascal language used by Embarcadero RAD Studio and Delphi. Object Pascal was chosen as the language because it is a very easy language to learn due to its very English-like keywords, and because it is structured and strongly-typed, allowing the resultant compiled applications to avoid run-time errors that can cause problems for un-typed languages like JavaScript (the target code of the compiler).

The following are the rules governing the basic structure of the language.

Character Set

Elevate Web Builder uses the Unicode character set for all language elements. Please see the Literals and Identifiers sections below for information on the restrictons to the allowed characters for both.

> **Warning**
> Although all Unicode characters are supported, certain double-wide characters in languages such as Chinese and Japanese cannot be displayed/edited properly in the Elevate Web Builder IDE and code editor.

Case-Sensitivity

Elevate Web Builder's language is not case-sensitive. Identifiers and other language keywords are always compared without considering case.

Whitespace and Line Breaks

Elevate Web Builder ignores any spaces or non-printable characters such as tabs or line feeds between identifiers or literals. Within string literals, any such characters are assumed to be included as part of the string itself. For example, the following code will cause the string literal to include a carriage return and line feed:

```
var
   MyStringVariable: String;
begin
   MyStringVariable:='This is a string literal with a
 carriage return and line feed included';
end;
```

Statement Terminator

The semicolon (;) is the code statement terminator character in Elevate Web Builder. It is used to indicate the ending of a statement, even if the statement spans more than one physical line:

```
begin
   if True then
      ShowMessage('It''s true !!!')
   else
      ShowMessage('It''s false !!!');
end;
```

In the above case, the extra line breaks are for formatting purposes only. However, you should always strive to format your code according to established formatting rules for the Object Pascal language, and such line breaks are very important for readability of your code.

Comments

Elevate Web Builder supports both single-line comments using two slashes (//) or multi-line comments using left and right braces ({}):

```
begin
   // This piece of code needs some work
   if (not True) then
      BlowUpTheApplication
   else
      begin
      { Whew, we avoided blowing up the application,
        so let's continue on a more reasonable path }
      HandleTheSpecialCase;
      end;
end;
```

Literals

Literal values are specified as follows:

Value Type	Example
Numbers	100 1200.42 -39.00
Boolean	True false
Strings	'This is a string literal' 'This is a '+' concatentated '+' string literal'
Characters	'a' #27
Arrays	['This','is','a','string','array','literal'] [100,2,45]
Class Instances/Methods	nil

Identifiers

An identifier is the name of any system-declared or user-declared object in an Elevate Web Builder application, such as units, constants, types, variables, or procedures/functions. Identifiers may begin with an underscore (_) or a letter (a-z, A-Z), and may contain an underscore, a letter, or a digit (0-9).

Reserved Words

The following are the list of reserved words in Elevate Web Builder. These words should not be used as identifiers:

```
abstract
and
array
as
async
begin
break
case
class
const
constructor
contains
continue
default
destructor
div
do
downto
else
end
except
exit
external
finalization
finally
for
```

```
function
if
implementation
inherited
initialization
interface
is
mod
not
object
of
on
or
out
override
private
procedure
program
property
protected
public
raise
read
record
repeat
shl
shr
then
to
try
type
unit
until
uses
var
virtual
while
with
write
xor
```

Syntax Diagrams

In the language reference syntax diagrams, angle brackets (<>) represent a language element and brackets ([]) represent an optional language element.

8.2 Defines

Elevate Web Builder supports basic compiler define functionality. Compiler defines are symbols used to conditionally include or exclude code in the compilation process, and can be tested at compile-time to make such a determination. Compiler defines are taken into account during the parsing phase of the compilation process.

Note
Elevate Web Builder defines a special DESIGN symbol automatically during compilation. If the code is being compiled for design-time use in the component library, then the DESIGN symbol will be defined. If the code is being compiled for run-time use, then the DESIGN symbol will not be defined. Component developers can test for this special DESIGN symbol to determine whether or not the code is being compiled for use at design-time. The standard component library included with Elevate Web Builder tests for this symbol in many different places.

Defining Symbols

You can create a compiler define using the following syntax:

```
{$DEFINE <Symbol>}
```

Once a symbol has been defined, it will be effective for the remaining code in the current unit, including any units that are referenced after the symbol was defined. Defining a symbol that is already defined does nothing.

Warning
Compiler defines are **not** nested. If a symbol is re-defined (it was already defined), and then un-defined, the result will be that the symbol will be **undefined**.

Un-Defining Symbols

You can remove a compiler define using the following syntax:

```
{$UNDEF <Symbol>}
```

Testing for Defined Symbols

To test whether a symbol has been defined, you can use the following syntax:

```
{$IFDEF <Symbol>}
```

```
// Include this code if the symbol is defined
[{$ELSE}]
// Include this code if the symbol is not defined (optional)
{$ENDIF}
```

To test whether a symbol has **not** been defined, you can use the following syntax:

```
{$IFNDEF <Symbol>}
// Include this code if the symbol is not defined
[{$ELSE}]
// Include this code if the symbol is defined (optional)
{$ENDIF}
```

An IFDEF or IFNDEF test must **always** be terminated with an ENDIF. The ELSE conditional branch is optional.

Example

The following is code from the standard component library that tests for the special DESIGN symbol to determine whether to use the WebDesign (IDE run-time) or the WebDOM (browser run-time) unit:

```
{$IFDEF DESIGN}
uses WebDesign, WebCore;
{$ELSE}
uses WebDOM, WebCore;
{$ENDIF}
```

8.3 Types

Elevate Web Builder supports most basic Object Pascal types, and these types are detailed below.

Exact Numeric Types

Exact numeric types are used when you wish to store a numeric value in its exact representation without accumulating rounding errors.

Type	Description
Integer	A 52-bit, signed integer value

Exact numeric literals use the minus (-) as the negative sign character, the plus (+) as the positive sign character, and scientific notation is not supported. In addition, hexadecimal literals can be specified by prefacing the hexadecimal value with the dollar sign ($).

The following are examples of exact numeric literals:

```
var
   MyInteger: Integer;
begin
   MyInteger := 100; // Assign 100 to the Integer variable
   MyInteger := $64; // Assign 100 as hexadecimal to the Integer variable
end;
```

Approximate Numeric Types

Approximate numeric types are used when you wish to store a numeric value in an approximate representation with a floating decimal point. Using approximate numeric types can cause rounding errors due to the fact that certain numbers such as 0.33 cannot be accurately represented using floating-point precision.

Type	Description
Double	A 64-bit, floating-point numeric value with a maximum precision of 16 digits.

Approximate numeric literals use the period (.) as the decimal point character, the minus (-) as the negative sign character, the plus (+) as the positive sign character, and scientific notation is supported via E (e or E) as the exponent character followed by a plus (+) or minus (-) character and the actual exponent value.

The following are examples of approximate numeric literals:

```
var
   MyDouble: Double;
```

```
begin
   MyDouble := -100.25; // Assign -100.25 to the Double variable
end;
```

String/Character Types

String types are used when you wish to store a character string of any length up to 2GB. String types always use the Unicode character set for the characters that comprise the string. Character types store a single character, and also use the Unicode character set.

Type	Description
String	A string value with a variable number of characters.
Char	A single character.

String literals use the single quote (') character to identify themselves as such. Any single quotes enclosed inside of the literal must be escaped by prefacing them with another single quote. In addition, single character constants may be specified using their literal value or by prefacing their ordinal character set position with the pound sign (#) character. To reference a specific character in a string, use the left and right brackets ([]) with the 1-based integer position of the character being referenced.

Note
Strings in Elevate Web Builder are immutable, meaning that they cannot be modified in-place by assigning new character values at specific positions in the string. They must always be copied and then assigned to a new string in order to be modified.

The following are examples of string/character literals:

```
var
   MyString: String;
   MyCharacter: Char;
begin
   MyString := 'This is a test'; // Assign "This is a test"
                                 // to the String variable
   MyString := #13+#10; // Assign a carriage return and
                        // linefeed to the String variable
   MyCharacter := MyString[2]; // Assign the second character
                               // from the String variable to
                               // the Char variable
end;
```

Date/Time Types

Date/time types are used when you wish to store a date, time, or date/time value. Date/time types are actually just integers, so they can be manipulated just like the Integer type.

Type	Description
DateTime	A date/time value containing the number of milliseconds since midnight on January 1, 1970.

Since date/time types are just integers, there isn't any literal representation of a date/time type.

Boolean Types

Boolean types are used to represent the values of True or False.

Type	Description
Boolean	A logical true/false value.

Boolean literals are expressed as the literals True and False (case-insensitive) or 1 and 0 for True and False, respectively.

The following are examples of boolean literals:

```
var
   MyBoolean: Boolean;
begin
   MyBoolean := False; // Assign False to the Boolean variable
end;
```

8.4 Operators

Elevate Web Builder supports most Object Pascal operators, and these operators are detailed below.

Boolean Operators

The following are the boolean operators in Elevate Web Builder, ordered by their operator precedence:

Operator	Description
not	Flips a boolean expression so that True becomes False, or vice-versa.
and	Returns True if both the left and right boolean expressions are True.
or	Returns True if either the left or right boolean expression is True.

Comparison Operators

The following are the comparison operators in Elevate Web Builder, ordered by their operator precedence:

Operator	Description
=	Returns True if both the left and right expressions are equal.
<>	Returns True if both the left and right expressions are not equal.
>	Returns True if the left expression is greater than the right expression.
>=	Returns True if the left expression is greater than or equal to the right expression.
<	Returns True if the left expression is less than the right expression.
<=	Returns True if the left expression is less than or equal to the right expression.
is	Returns True if the left expression is an instance of the class type specified in the right expression.

Arithmetic Operators

The following are the arithmetic operators in Elevate Web Builder, ordered by their operator precedence:

Operator	Description
not	Returns an integer that represents the inverse of all bits in the right integer expression.
or	Returns an integer that represents all set bits in the left and right integer expressions.
xor	Returns an integer that represents all set bits in either the left or right, but not both, integer expressions.
and	Returns an integer that represents all bits that are set in both the left and right integer expressions.
*	Multiplies the left numeric expression by the right numeric expression.
/	Divides the left numeric expression by the right numeric expression.
div	Divides the left integer expression by the right integer expression.
-	Subtracts the right numeric expression from the left numeric expression.
+	Adds the right numeric expression to the left numeric expression.
mod	Returns the remainder derived from dividing the left numeric expression by the right numeric expression.
shl	Returns the left integer expression shifted to the left by the number of bits specified by the right integer expression.
shr	Returns the left integer expression shifted to the right by the number of bits specified by the right integer expression.

String Operators

The following are the string operators in Elevate Web Builder, ordered by their operator precedence:

Operator	Description
+	Concatenates the right string expression to the left string expression.

8.5 Statements

Elevate Web Builder supports most Object Pascal statements, and these statements are detailed below. Please see the Function and Procedure Implementations topic for information on how statements are actually used in function and procedure code blocks.

Assignment Statement

```
<Variable> := <Type-Compatible Expression>;
```

The assignment statement uses the assignment operator (:=) to assign a value from a type-compatible expression on right-hand side of the assignment operator to the variable on the left-hand side of the operator.

The following example illustrates the use of the assignment statement:

```
var
   MyInteger: Integer;
   MyString: String;
begin
   MyInteger := (100 * MyIntegerConstant);
   MyString := 'This is a test';
end;
```

If Statement

```
if <Boolean Expression> then
   <Code Block>
[else if <Boolean Expression>
   <Code Block>]
[else
   <Code Block>];
```

The if statement is used to provide conditional execution based upon one or more Boolean expressions. When any of the Boolean expressions specified in the if or else if clauses evaluates to True, then the block of statements is executed. The else clause is used to specify that the if none of the Boolean expressions evaluate to True, then the statement block specified for the else clause should be executed.

The following example illustrates the use of the if statement:

```
var
   MyBoolean: Boolean;
begin
   MyBoolean:=False;
   if MyBoolean then
      ShowMessage('This will never execute')
   else
      ShowMessage('This will always execute');
end;
```

Case Statement

```
case <Expression> of
   <Expression>[, <Expression>}:
      <Code Block>;
   [<Expression>[, <Expression>}:
      <Code Block>;]
   [else
      <Code Block>;]
   end;
```

The case statement is used to provide conditional execution based upon one or more expression comparisons. The expression directly after the case clause is compared against each expression specified before the colon (:). If any of the expression comparisons are equal, then the block of statements directly after the colon is executed. The else clause is used to specify that if none of the expression comparisons are equal, then the statement block specified for the else clause should be executed.

The following example illustrates the use of the case statement:

```
var
   MyString: String;
begin
   MyString:='Hello World';
   case MyString of
      'Hello':
         ShowMessage('Hello');
      'World':
         ShowMessage('World');
      else
         ShowMessage('None of the above');
      end;
end;
```

While Statement

```
while <Boolean Expression> do
      <Code Block>;
```

The while statement is used to provide a looping construct based upon a Boolean expression comparison. The Boolean expression directly after the while clause is compared before every execution of the block of statements. If the Boolean expression evaluates to False, then the loop is terminated and execution will continue on the statement after the block of statements that belong to the while statement.

The following example illustrates the use of the while statement:

```
var
   MyBoolean: Boolean;
begin
   MyBoolean:=True;
   while MyBoolean do
      begin
      ShowMessage('Still looping...');
      if MyBoolean then
         MyBoolean:=False;
      end;
end;
```

Repeat Statement

```
repeat
      <Code Block>;
until <Boolean Expression>;
```

The repeat statement is used to provide a looping construct based upon a Boolean expression comparison. The Boolean expression directly after the util clause is compared after every execution of the block of statements. If the Boolean expression evaluates to True, then the loop is terminated and execution will continue on the statement after the block of statements that belong to the repeat statement.

The following example illustrates the use of the repeat statement:

```
var
   MyBoolean: Boolean;
begin
   MyBoolean:=False;
   repeat
      begin
      ShowMessage('Still looping...');
      if (not MyBoolean) then
         MyBoolean:=True;
      end;
   until MyBoolean;
end;
```

For Statement

```
for <Integer Value Assignment> to|downto <Integer Expression> do
      <Code Block>;
```

The for statement is used to provide a looping construct based upon an incrementing or decrementing integer value comparison. The loop is seeded with an an integer value assignment that is an assignment statement without a semicolon statement terminator (;). If the to clause is used, then the integer value will be incremented by one for every iteration of the loop, and if the downto clause is used, then the integer value will be decremented by one for every interation of the loop. The integer expression after the to or downto clause serves as the terminator for the loop. Once the integer value is equal to the value of the specified integer expression, the loop is terminated and execution will continue on the statement after the block of statements that belong to the for statement.

The following example illustrates the use of the for statement:

```
var
   MyInteger: Integer;
   MyString: String='Hello world';
begin
   for MyInteger:=1 to Length(MyString) do
      ShowMessage('Character is '+MyString[MyInteger]+'...');
end;
```

Break Statement

```
break;
```

The break statement is used to unconditionally break out of any looping statement (while, repeat, or for). Any time a break statement is encountered, the loop is terminated and execution will continue on the statement after the block of statements that belong to the looping statement.

The following example illustrates the use of the break statement:

```
var
   MyInteger: Integer;
   MyString: String='Hello world';
begin
   for MyInteger:=1 to Length(MyString) do
      begin
      ShowMessage('Character is '+MyString[MyInteger]+'...');
      if MyString[MyInteger]='w' then
         break;
      end;
end;
```

Continue Statement

```
continue;
```

The continue statement is used to unconditionally stop executing any and all remaining statements in the block of statements for a looping statement (while, repeat, or for) and return to the top of the looping statement.

The following example illustrates the use of the continue statement:

```
var
   MyBoolean: Boolean;
begin
   MyBoolean:=True;
   while MyBoolean do
      begin
      ShowMessage('Still looping...forever');
      continue;
      if MyBoolean then
         MyBoolean:=False;
      end;
end;
```

Exit Statement

```
exit;
```

The exit statement is used to unconditionally stop executing any and all remaining statements in a function or procedure, and leave the function or procedure.

With Statement

```
with <Class Instance> do
   <Code Block>;
```

The with statement is used to introduce a class instance into the scope of the block of statements specified after the do clause. This is useful when one needs to reference several different properties or methods of the class instance and would like to avoid repeatedly typing the same class instance reference.

The following example illustrates the use of the with statement:

```
var
   MyInstance: TMyClass;
begin
   MyInstance:=TMyClass.Create;
   with MyInstance do
      begin
      MyIntegerProperty:=100;
      MyStringProperty:='Hello world';
      end;
   MyInstance.Free;
end;
```

Code Blocks

One or more statements in succession is referred to as a code block. If more than one statement is included in a code block, then the code block must be expressed with the **begin** and **end** keywords:

```
begin
 <Statement>;
 [<Statement>];
end;
```

For example, the following for loop can be expressed without the begin and end keywords because its code block only consists of a single statement:

```
var
   MyInteger: Integer;
   MyString: String='Hello world';
   MyOtherString: String='';
begin
   for MyInteger:=1 to Length(MyString) do
      MyOtherString := MyOtherString + MyString[MyInteger];
end;
```

Code blocks can be nested as many levels deep as necessary.

Statement Termination

One of the most confusing aspects of the Object Pascal language used by Elevate Web Builder is how to decide when to terminate a statement. Normally, all statements must be terminated with the statement terminator character (;) when the statements are used by themselves. For example, the following assignment statement is terminated in a normal fashion because it is used by itself in a code block:

```
var
   MyString: String;
begin
   MyString := 'This is a test';
end;
```

However, when a statement is embedded within another container statement, such as an if statement, the statement terminator may not be needed. For example, the same assignment statement used above would look like this when used in an if statement:

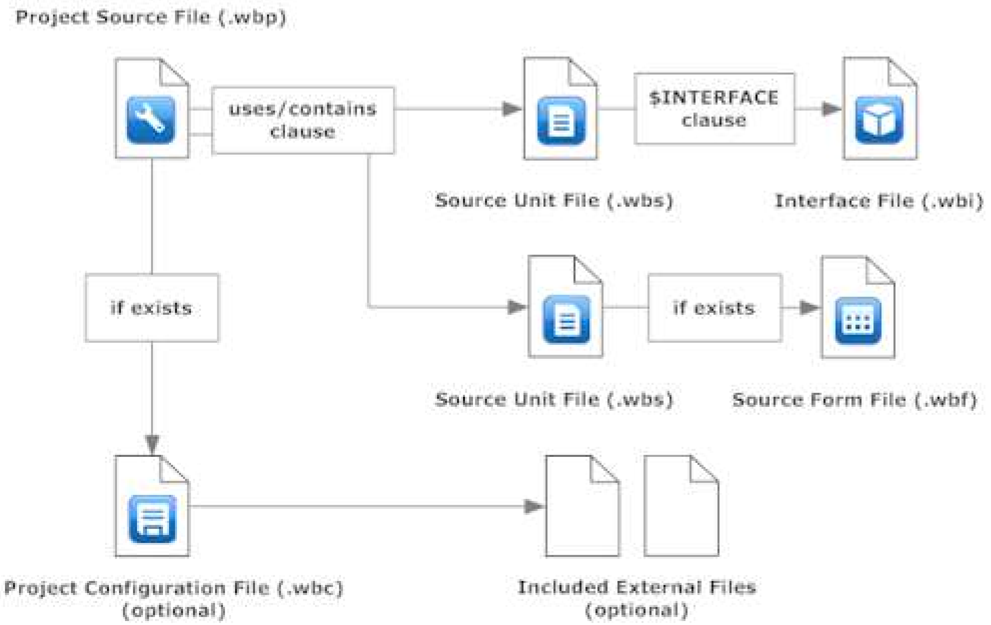
```
var
   MyString: String;
begin
   if MyParameter then
      MyString := 'This is a test'
   else
      MyString := 'This is not a test';
end;
```

The first assignment statement does not require a statement terminator because it is considered part of the if statement, whereas the second assignment statement has a statement terminator because it is used in the else clause of the if statement.

The exception to this rule occurs when a statement is used within a code block. Statements **always** require a statement terminator when used in a code block.

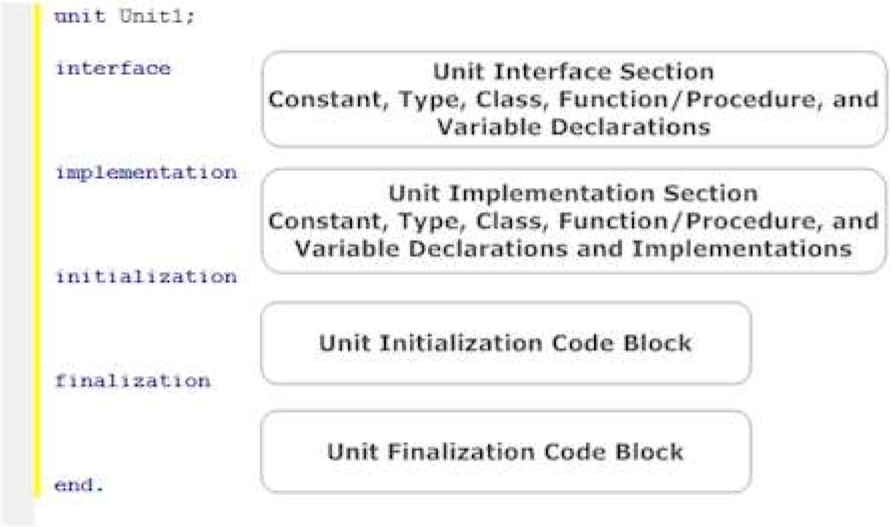
8.6 Units

The Elevate Web Builder language uses the source unit as the basis for all code in an application. As discussed in the General Architecture topic, every application has the following basic structure:



Elevate Web Builder Application Structure

Every source unit in an application has the following structure:


unit Unit1;
interface
Unit Interface Section
Constant, Type, Class, Function/Procedure, and Variable Declarations
implementation
Unit Implementation Section
Constant, Type, Class, Function/Procedure, and Variable Declarations and Implementations
initialization
Unit Initialization Code Block
finalization
Unit Finalization Code Block
end.

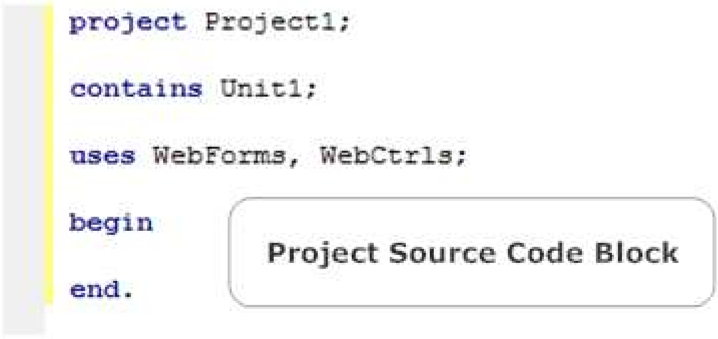
Elevate Web Builder Source Unit Structure

Every source unit must begin with the keyword **unit** followed by the name of the unit (without file extension) and the statement terminator (;). In addition, every source unit must end with the **end** keyword followed by a period (.).

The interface and implementation section keywords are also required. The initialization and finalization code blocks are both optional, and one can be specified without the other.

Project Source File

The project source (.wbp) of an application uses a format similar to a normal unit, but with some key changes:



```
project Project1;

contains Unit1;

uses WebForms, WebCtrls;

begin

end.
```

Elevate Web Builder Project Source Structure

The key changes are:

- The project source file begins with the **project** keyword instead of the **unit** keyword.
- The project source file has a contains clause. The contains clause is just like a uses clause but also determines which units are considered project units, as opposed to simply units referenced by the project. The IDE uses this information to determine which units should be shown as part of the project in the Project Manager.
- The project source file only contains a single uses clause.
- The project source file does not contain interface, implementation, initialization, and finalization sections. You can add constant, type, class, function/procedure, and variable declarations between the uses clause and the main code block, but this is strictly optional.
- The project source file contains a single code block (begin..end) that is the first code to be executed when the application starts.

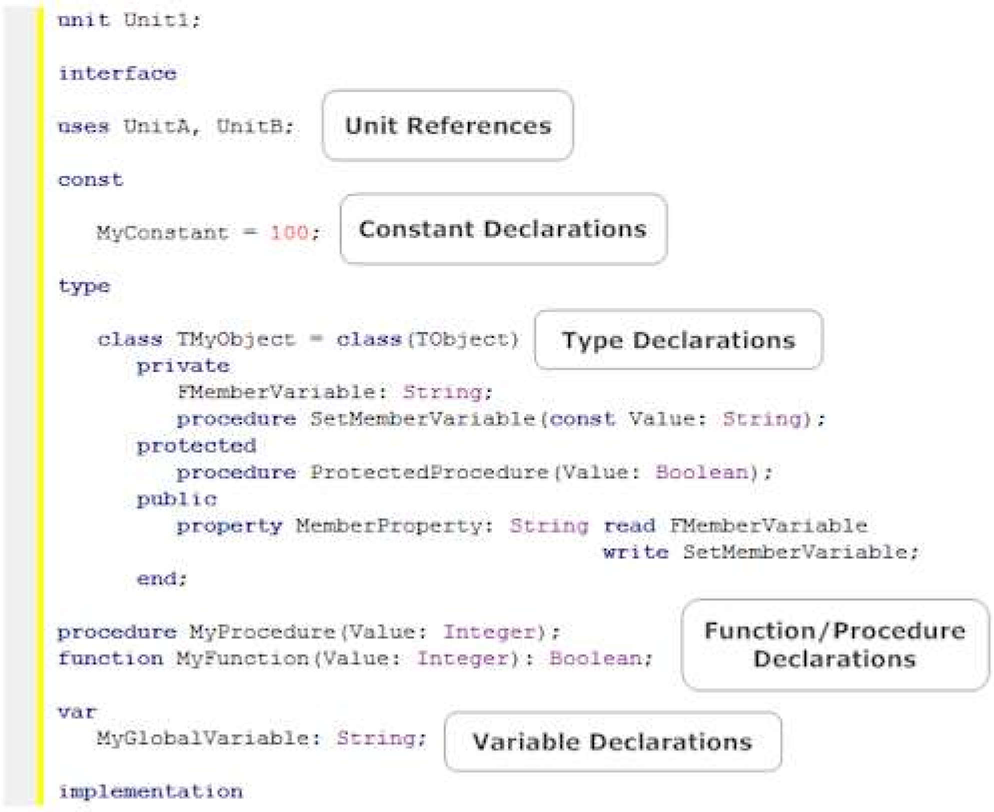
Interface and Implementation Sections

The interface and implementation sections are very similar in structure. The main differences are in the scope (visibility) of each section (private or public) and whether the section can only contain declarations and not implementation code:

- All declarations and code in the implementation section are private to the source unit and cannot be referenced by other souce units. All declarations in the interface section are public to both the current source unit and other source units. For example, if you were to declare the TMyClass class in the implementation section of UnitA, then even if UnitB included a UnitA reference in its uses clause (interface or implementation, it doesn't matter), the TMyClass class declaration would still not be "visible" to UnitB. Please see the Scope topic for more information.

- The interface section can only contain declarations, whereas the implementation section can also include implementation code for functions, procedures, and methods of classes (functions and procedures declared in classes).

Both the interface and implementation sections have the following elements in common:



```
unit Unit1;

interface

uses UnitA, UnitB;

const

   MyConstant = 100;

type

   class TMyObject = class(TObject)
      private
         FMemberVariable: String;
         procedure SetMemberVariable(const Value: String);
      protected
         procedure ProtectedProcedure(Value: Boolean);
      public
         property MemberProperty: String read FMemberVariable
                                         write SetMemberVariable;
      end;

procedure MyProcedure(Value: Integer);
function MyFunction(Value: Integer): Boolean;

var
   MyGlobalVariable: String;

implementation
```

Elevate Web Builder Unit Section Structure

Note
None of the various section elements are required. In fact, one can have a valid source unit that includes nothing but the unit name, the interface and implementation clauses, and the end keyword. It would be of little use, but it would still be valid.

Uses Clause

The uses clause is a comma-separated list of source unit names (*.wbs, but specified without the file extension). This clause tells the compiler which source units are being referenced by the code in the interface or implementation sections.

Const Clause

The const clause is used to declare constants. Please see the Constant Declarations topic for more information on declaring constants. You can declare as many constants as you wish within the same const clause.

Type Clause

The type clause is used to declare types. Please see the Type Declarations topic for more information on declaring types. You can declare as many types and classes as you wish within the same type clause.

Var Clause

The var clause is used to declare global variables. Please see the Variable Declarations topic for more information on declaring variables. You can declare as many variables as you wish within the same var clause.

Function and Procedure Declarations

You can include function and procedure declarations anywhere in an interface or implementation section. However, normally one would only include a function or procedure declaration in the interface section. Since a function or procedure is actually implemented in the implementation section, there is no purpose to declaring the function or procedure there twice. Please see the Function and Procedure Declarations topic for more information on declaring functions and procedures.

Initialization and Finalization Sections

The initialization and finalization sections are used to add code blocks for initializing variables in a unit at application startup and for freeing any resources acquired during execution at application shutdown. For example, in the Elevate Web Builder component library, an instance of the TApplication component is automatically created and freed in the initialization and finalization code blocks of the WebForms unit:

```
initialization
   Application:=TApplication.Create(nil);
finalization
   Application.Free;
   Application:=nil;
end.
```

The order in which the initialization and finalization code blocks are executed is determined by the order of the unit references in the uses clause of the project source file, as well as the order of the unit references in the uses clauses of the interface and implementation sections of each referenced unit.

The initialization order is as follows:

- The units in the project source file's uses clause are initialized in the order in which they are specified.

- The units in each referenced unit's uses clause are initialized in the order in which they are specified. The units in the uses clause in the interface section of the source unit are initialized first, followed by the units in the uses clause in the implementation section of the source unit.

- After all referenced units have been initialized, the initialization code block for the current unit is executed.

The finalization order is the reverse of the above:

- The units in the project source file's uses clause are finalized in the reverse order in which they are specified.

- The finalization code block for the current unit is executed.

- The units in each referenced unit's uses clause are finalized in the reverse order in which they are specified. The units in the uses clause in the implementation section of the source unit are finalized first, followed by the units in the uses clause in the interface section of the source unit.

8.7 Constant Declarations

Constant declarations must be in the following format:

```
<Constant Declaration>;
[<Constant Declaration>;]

<Constant Declaration> =

   <Constant Name> = <Expression>
```

The <Constant Name> element must follow the rules for identifiers covered in the Introduction topic, and each constant declaration must be terminated with a semicolon statement terminator (;).

The <Expression> element can be any valid expression that does not contain any variable, function, or procedure references. In other words, the expression can only contain other constant references, or literal expressions. Examples of literal expressions can also be found in the Introduction topic.

8.8 Type Declarations

Type declarations must be in the following format:

```
<Type Declaration> | <Function/Procedure Type Declaration> | <Method Type
      Declaration> | <Class Declaration>;
[<Type Declaration> | <Function/Procedure Type Declaration> | <Method Type
      Declaration> | <Class Declaration>;]

<Type Declaration> =

   <Synonym Type Name> = [type] <Type Name>;

<Function/Procedure Type Declaration> =

   <Function/Procedure Type Name> = function|procedure ([<Parameters>])[:
      <Type Name>];

<Method Type Declaration> =

   <Method Type Name> = function|procedure ([<Parameters>])[: <Type Name>] of
      object;

<Class Declaration> =

   <Class Name> = class [(<Ancestor Class Name>)]
      [<Private Class Members>]
      [<Protected Class Members>]
      [<Public Class Members>]
      end;
```

The <Synonym Type Name>, <Type Name>, <Method Type Name>, <Class Name>, and <Ancestor Class Name> elements must follow the rules for identifiers covered in the Introduction topic, and each type or class declaration must be terminated with a semicolon statement terminator (;).

A synonym type declaration is useful for situations where one wants to use a specific name for a generic type such as an Integer, String, or Boolean. For example, in the Elevate Web Builder component library, the following type declaration can be found in the WebUI source unit:

```
TColor = type Integer;
```

The optional **type** keyword found before the Integer type name above is used to specify that the declared type (TColor) should be considered unique for design-time type distinction purposes, and essentially creates a new Integer type called TColor. The Elevate Web Builder IDE design-time environment uses this information to determine which property editors to show for certain types.

Note

The synonym type retains type compatibility with the type name that it is associated with. For example, with the above declaration the compiler will still allow you to use an Integer in any expression where a TColor type is required.

A function/procedure type declaration is used to declare a function/procedure reference type. Function/procedure references point to a function/procedure implementation, and are used to treat functions/procedures as data. The Elevate Web Builder compiler automatically figures out when a function/procedure reference is being assigned to a variable declared with such a reference type, and when a variable that is declared with such a reference type is being used to call the function/procedure reference contained in the variable. You can only assign references to functions/procedures that have the same signature as the target variable's function/procedure reference type. For example, this assignment is not valid:

```
type

TFuncRef = function (Value: Integer): Integer;  // Returns Integer

implementation

function DoSomething(Value: Integer): Boolean;  // Returns Boolean
begin
   Result:=(Value=100);
end;

procedure DoSomethingElse;
var
   FuncRef: TFuncRef;
begin
   FuncRef:=DoSomething;  //  This will cause a compiler error !!!!
   ShowMessage(IntToStr(FuncRef(100)));
end;
```

A method type declaration is used to declare a method reference type. A method reference type is like a function/procedure reference type, except that it also includes the class instance to be used when calling the method contained within a variable declared with a method reference type. For more information on method reference type declarations, please see the Events topic.

A class declaration can include an ancestor class name, if applicable. If one is declaraing a class that inherits from the base TObject class, then the ancestor class name does not need to be specified. For more information on class declarations, please see the Classes topic.

8.9 Variable Declarations

Variable declarations must be in the following format:

```
<Variable Declaration>;
[<Variable Declaration>;]

<Variable Declaration> =

   <Variable Name> [,<Variable Name>]: <Type Name> = <Default Expression>
```

The <Variable Name> and <Type Name> elements must follow the rules for identifiers covered in the Introduction topic, and each variable declaration must be terminated with a semicolon statement terminator (;).

The <Default Expression> element can be any valid expression that does not contain any variable, function, or procedure references. In other words, the expression can only contain other constant references, or literal expressions. Examples of literal expressions can also be found in the Introduction topic. The <Default Expression> element is used to initialize the variable to a specific value. This is useful in ensuring that a variable is not in an uninitialized state when it is referenced in code.

Warning

If you don't set a default expression for a variable declaration and do not assign a value to the variable, then the value of the variable is undetermined. You should always make sure that such global variable declarations are initialized properly through a default expression or assignment. This does not, however, apply to variables declared within a class. Please see the Variables topic for more information on declaring variables in a class.

8.10 Function and Procedure Declarations

Function and procedure declarations must be in the following format:

```
<Function Declaration> | <Procedure Declaration>;
[<Function Declaration> | <Procedure Declaration>;]

<Function Declaration> =

   function <Function Name> [<Parameters>]: <Type Name>

<Procedure Declaration> =

   procedure <Procedure Name> [<Parameters>]

<Parameters> =

  (<Parameter Declaration>[; <Parameter Declaration>])

<Parameter Declaration> =

   [const] <Parameter Name> [,<Parameter Name>]: <Type Name>
```

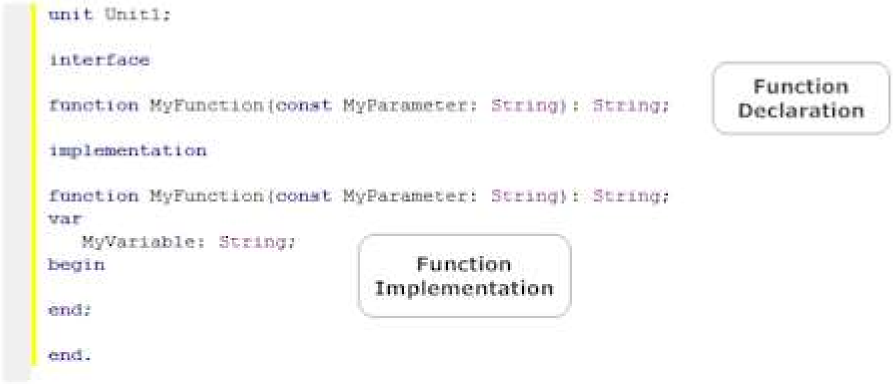
The <Function Name>, <Procedure Name>, <Parameter Name>, and <Type Name> elements must follow the rules for identifiers covered in the Introduction topic, and each function or procedure declaration must be terminated with a semicolon statement terminator (;).

The only difference between a function and procedure is that a function returns a result value, whereas a procedure does not. This means that a function can be used in an expression like a variable or constant, but a procedure can only be used like a statement.

Please see the Function and Procedure Implementations topic for more information on implementing functions and procedures.

8.11 Function and Procedure Implementations

Functions and procedures constitute the actual functionality of an Elevate Web Builder unit. The implementation of functions and procedures is done in the implementation section of a unit:



```
unit Unit1;

interface

function MyFunction(const MyParameter: String): String;

implementation

function MyFunction(const MyParameter: String): String;
var
   MyVariable: String;
begin

end;

end.
```

Function/Procedure Implementation

The implementation of a function or procedure consists of a variable declaration block, if necessary, followed by a code block:

```
procedure MyProcedure;
var
   MyVariable: String;
   MyOtherVariable: Integer;
begin
   // Code block
end;
```

Note
The implementation of a function/procedure repeats the same declaration of the function or procedure name followed by the parameters (if present) and return type (for functions). If the implementation does not use the exact same declaration, then the compiler will issue an error.

The variable declarations follow the same declaration rules as the **var** clause in a unit. Please see the Variable Declarations topic for more information.

A code block in Elevate Web Builder consists of a **begin** keyword followed by a series of statements and an **end** keyword. The code block that is used in a function or procedure implementation is always terminated with a statement terminator (;). However, code blocks can be nested within other statements, such as conditional if statements. In such cases, please refer to the documentation on the statement to determine if a statement terminator is necessary after the end keyword at the end of a code block.

Returning Results From a Function

In Elevate Web Builder, procedures do not return a result, whereas functions do. Every function has an implicit **Result** variable that can be assigned a value that is returned as the result of the function:

```
function MyFunction(const MyParameter: String): String;
begin
   Result := 'The parameter value is ' + MyParameter;
end;
```

The type of the Result variable is determined by the type declaration for the function's return value.

8.12 Enumerations

An enumeration is a collection of symbols used to represent a specific set of values. An enumeration must be declared as a specific type and, because they are typed, enumerations offer the additional benefit of preventing improper symbolic values that aren't part of the enumeration from being used anywhere that the enumeration type is required.

An enumeration is declared as follows:

```
<EnumerationName> = (<Member Name>[,<Member Name>]);
```

For example, the following enumeration type declaration is from the WebUI unit in the Elevate Web Builder component library and specifies the various cursor types that can be used in a UI element:

```
TCursor = (crAuto,crCrossHair,crDefault,crHelp,crMove,crPointer,
           crProgress,crSizeNESW,crSizeNS,crSizeNWSE,crSizeWE,
           crText,crWait);
```

Note
 Internally, enumerations are handled as integers by the compiler, and you can cast enumerations as integers and integers as enumerations.

8.13 Arrays

An array is a collection of values, all of the same type. The values in an array are referred to as the array's elements. Arrays in Elevate Web Builder are dynamic, meaning that their length is not specified during their declaration and can be increased or decreased at run-time, as necessary.

Array Declarations

Arrays are declared by prefacing the type name of the array with the keywords **array of**:

```
array of <Type Name>
```

For example, to declare a Boolean array variable, one would use the following declaration:

```
var
   MyBooleanArray: array of Boolean;
```

Please see the Variable Declarations topic for more information on declaring variables.

Getting and Setting the Length of an Array

To get the length of an array, use the Length function. Likewise, use the SetLength function to set the length of an array:

```
var
   MyArray: array of Integer;
begin
   SetLength(MyArray,10);
   ShowMessage(Length(MyArray)); // Displays the value 10
end;
```

Referencing an Array Element

Each element in an array can be accessed via its 0-based ordinal position. Square brackets directly after the array variable or parameter name are used to reference an element in an array. For example, to access the 3rd element in an array, one would use the following construct:

```
MyArray[2]
```

Warning
If you try to access an element that does not exist because it is beyond the length of the array, you will cause a run-time error. Also, you must declare an array variable with a default value (see below), or use the SetLength function to set the length of an array, before attempting to reference any of the array elements. Failure to do so will result in a run-time error.

Array Constants

Just like any other variable, array variables can be initialized to a default value by specifiying an array constant as the default value in the variable declaration. This is done by enclosing a comma-delimited list of array elements in square brackets ([]):

```
[<Element> ,<Element>...]
```

In the following example the MyBooleanArray variable will be initialized to an array of Boolean elements that has a length of 3 and consists of elements that are True, False, and True, respectively:

```
var
   MyBooleanArray: array of Boolean = [True, False, True];
```

In addition to variable defaults, array constants can be used to pass array values to the parameters of functions or procedures. For example, if we want to call a procedure declared as follows:

```
function ListStrings(Value: array of String): String;
var
   I: Integer;
begin
   Result := '';
   for I := 0 to Length(Value) - 1 do
      begin
      if (I > 0) then
         Result := Result + ', ';
      Result := Result + Value[I];
      end;
end;
```

We could do so by simply passing a constant array to it, as follows:

```
ShowMessage(ListStrings(['These','are','some','words']);
```

Note
The values specified for the elements in an array constant must be type-compatible with the declared type of the array.

8.14 Classes

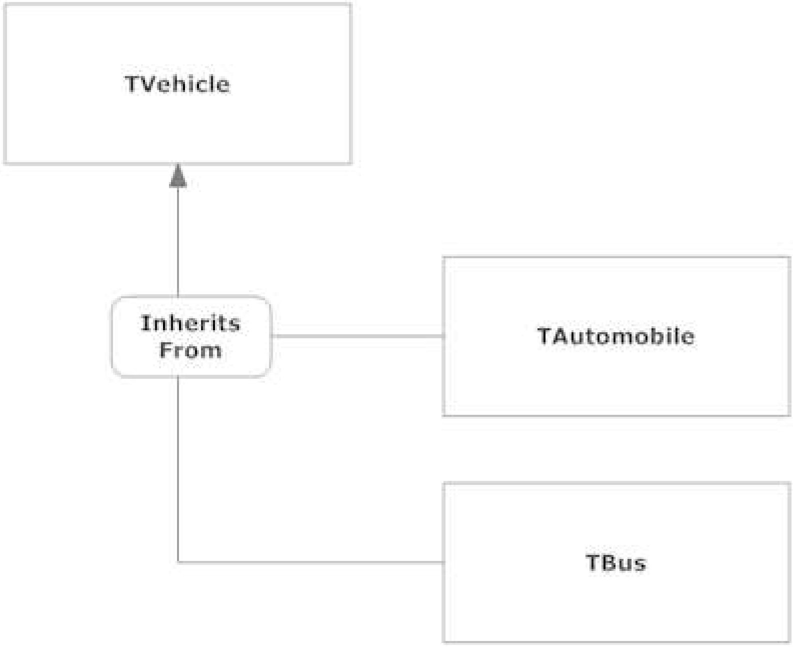
The Object Pascal language used by Elevate Web Builder is an object-oriented language that allows one to define classes that represent objects with their own data (variables and properties) and behaviors (functions and procedures).

> **Note**
> Functions and procedures that are declared in a class are referred to as "methods". For the rest of this topic the term "methods" will be used to represent the functions and procedures declared in a class.

Classes are useful because they offer:

- **Encapsulation**: Both data and behaviors are combined into one logical construct, and data that is internal to the class can be hidden so that only the class itself can access it. Also, properties can be defined that offer a specific interface to the data contained in a class, thus avoiding exposing internal data directly or requiring that all access to the data be done through methods.

- **Inheritance**: Classes can descend from other classes and inherit the functionality of the ancestor class in the process, thus forming what is termed a "class hierarchy". There is no limit to the depth of such a hierarchy. In addition, the functionality of ancestor class(es) can be overridden in descendant classes in order to supplement or completely replace the base functionality.



Class Hierarchy Example

Elevate Web Builder only supports single inheritance. This means that each class can only descend from one class, and there is only one single path between an ancestor class and a descendant class.

Class Declarations

Before a class can be used, it must be declared in the type section of a unit. A class declaration consists of the following:

```
<Class Name> = class [(<Ancestor Class Name>)]
   [<Private Class Members>]
   [<Protected Class Members>]
   [<Public Class Members>]
   [<Published Class Members>]
   end;
```

Note
 To keep with the traditional coding style of Object Pascal, all <Class Name> specifications in Elevate Web Builder should normally begin with a "T" prefix (stands for "Type").

If you do not specify an <Ancestor Class Name>, then the class will inherit from the system-defined TObject class (see below).

The private, protected, public, and published designations specify the scope of the class members. The scope of the class members determine their visibility to descendant class declarations, as well as any code that uses an instance of the class. In addition, the published scope determines which properties are visible at design-time in the IDE and which properties are streamable using the persistence functionality in the component library. Please see the Scope topic for more information on the various class member scope designations.

TObject

As expected, any class hierarchy must start with a base class. In Elevate Web Builder, that class is the TObject class. The declaration for the base TObject class is as follows:

```
external system TObject = class
   public
      constructor Create; virtual;
      destructor Destroy; virtual;
      class procedure Free;
      class function ClassType: TClass;
      class function ClassName: String;
      class function ClassParent: TClass;
   end;
```

The Create method is the class constructor and the Destroy method is the class destructor. The constructor method is called when a class instance is created, and the destructor method is called when a class instance is freed. Please see the Methods topic for more information on constructors and destructors.

The ClassType method is a class method that returns the class type as a result. This method is useful in situations where you need to interrogate the class type of either a class or a class instance.

The ClassName method is a class method that returns the class name as a result. The Elevate Web Builder component library uses this method a lot in order to link control classes to specific interfaces.

The ClassParent method is a class method that returns the parent class type as a result. This information can be used to determine the ancestry of a class.

> **Note**
> The above Class* methods are class methods, which means that they are static methods that can operate on both classes and class instances. Please see the Methods topic for more information on class methods.

Class Members

The class members in a class declaration are the various variables, properties, methods, and events that define the data and behaviors of the class. Class members can be declared in any order within a specific scope in a class declaration.

Please use the following links to get more information on each type of class member:

Variables
Methods
Properties
Events

8.15 Variables (In Classes)

Variables are declared in a class just like they are declared in a unit or function/procedure. Please see the Variable Declarations topic for more information.

> **Note**
> As a code convention, variable declarations in classes are normally prefaced with an "F" to distinguish them from other variables and properties. The "F" stands for "Field", but this manual will refer to them as variables and not "fields".

If you don't set a default expression for a variable declaration in a class declaration, then the variable will be automatically initialized to the appropriate value for the type when an instance of the class is created:

Type	Initial Value
String	''
Char	#0
Integer Double	0
Enumerated Type	Lowest Member Value
Boolean	False
DateTime	0
Object Array Method	nil

Class Variables

Class variables are special types of variables that are sometimes referred to as "static" variables. They can be referenced from class instances and also directly from class references where no instance of the class exist, and are useful for storing data that doesn't change between instances of a class. Class variables can only be modified by class methods or class properties. Please see the Methods and Properties topics for more information on class methods and properties.

Class variables are declared by prefacing a variable declaration with the **class** keyword. For example, the following class declaration includes a class variable that keeps track of how many instances of the class have been created:

```
TMyClass = class
   private
      class FCreateCount: Integer;
   public
      constructor Create; override;
      class property CreateCount: Integer read FCreateCount;
   end;
```

```
implementation

constructor TMyClass.Create;
begin
   inherited Create;
   Inc(FCreateCount);
end;
```

The CreateCount class property above, and subsequently the FCreateCount class variable, could be accessed in two different ways. The first way is by referring to the CreateCount class property for an instance of the TMyClass class:

```
procedure ShowCreateCount;
var
   TempInstance: TMyClass;
begin
   TempInstance:=TMyClass.Create;
   try
      window.alert(IntToStr(TempInstance.CreateCount));  // window class is
      in WebDOM unit
   finally
      TempInstance.Free;
   end;
end;
```

The second way is by using a direct class reference:

```
procedure ShowCreateCount;
begin
   window.alert(IntToStr(TMyClass.CreateCount));
end;
```

The second way is easiest when you don't have an instance of the class available.

One of the most significant benefits of class variables is that only one instance of each class variable ever exists, thus saving memory. They are also very useful for implementing singleton instances of classes and implementing namespaces for code that otherwise would use normal functions and procedures declared outside of a class.

8.16 Methods

Methods are simply functions and procedures that are declared as part of a class declaration. They are declared in the same way as functions and procedures that are declared in a unit. Please see the Function and Procedure Declarations topic for more information on the proper syntax. However, methods offer three additional keywords: **virtual**, **abstract**, and **override**.

Virtual Methods

The **virtual** keyword allows you to specify whether or not a method can be overridden by descendant classes. Virtual methods form the basis of inheritance in an object-oriented language because they allow the developer to supplement or replace existing functionality in an ancestor class with functionality that is more specific to the current class. For example, consider the following example class declarations and implementations:

```
interface

   TVehicle = class
      protected
         function GetNumWheels: Integer; virtual;
      public
         property NumWheels: Integer read GetNumWheels;
      end;

   TTruck = class(TVehicle)
      protected
         function GetNumWheels: Integer; override;
      end;

implementation

function TVehicle.GetNumWheels: Integer;
begin
   Result := 4;
end;

function TTruck.GetNumWheels: Integer;
begin
   Result := 10;
end;
```

As you can see, the default value returned from the GetNumWheels method is 4. But, because the GetNumWheels method is virtual, descendant classes like the TTruck class can override the method to provide a different result that is accurate for the type of vehicle being represented by the class.

Note

A virtual method does not have to be present in the immediate ancestor class in order for it to be overridden. You can override **any** virtual method that exists in **any** ancestor class, no matter how far removed it is from the class being declared. Also, once a method is declared as virtual, it is always virtual and capable of being overridden by a descendant class.

In addition, you can use the **abstract** keyword to specify that the virtual method isn't actually implemented in the current class, but rather **must** be implemented by descendant classes. Using the above example, the base TVehicle class could be declared and implemented as follows instead:

```
interface

   TVehicle = class
      protected
         function GetNumWheels: Integer; virtual; abstract;
      public
         property NumWheels: Integer read GetNumWheels;
      end;

   TTruck = class(TVehicle)
      protected
         function GetNumWheels: Integer; override;
      end;

implementation

function TTruck.GetNumWheels: Integer;
begin
   Result := 10;
end;
```

Note

Any classes that contain abstract methods cannot be created directly. Any attempt to do so will cause a compiler error.

If you want to augment, or add to, the functionality present in the virtual method of an ancestor class, then you can use the **inherited** keyword in the implementation of the descendant class method to do so. For example, in this class hierarchy the TCar class defines a GetAvailableColors method that returns a comma-delimited list of the default colors that a car is available in for the car manufacturer. The descendant TPython class that represents a sports car overrides the GetAvailableColors method to specify additional colors that the sports car is available in:

```
interface

   TCar = class
      protected
         function GetAvailableColors: String; virtual;
      public
         property AvailableColors: String read GetAvailableColors;
      end;

   TPython = class(TCar)
      protected
         function GetAvailableColors: String; override;
      end;

implementation

function TCar.GetAvailableColors: String;
```

```
begin
   Result := 'Red, Black, White';
end;

function TPython.GetAvailableColors: String;
begin
   Result := inherited GetAvailableColors + ', Silver';
end;
```

Overloaded Methods

Overloaded methods are methods that have more than one declaration in a class and each declaration has the same name but different parameters. This is very useful for situations where a method may need to be called using different parameters. For example, consider the following class declaration:

```
TCustomers = class
   public
      procedure Delete(ID: Integer);
      procedure Delete(const Name: String);
   end;
```

In this example, the TCustomers class has overloaded the Delete method so that it can be called with either an integer customer ID or a string customer name.

Although default parameters are usually easier, overloaded methods can also be used to implement optional parameters. For example, the following class allows its Add method to be called with an ID, a name, or both:

```
TCustomers = class
   public
      procedure Add(ID: Integer);
      procedure Add(ID: Integer; const Name: String);
      procedure Add(const Name: String);
   end;
```

Note
Other variants of Object Pascal require that you use the **overload** keyword to indicate that a method is overloaded. Elevate Web Builder does not use the overloaded keyword because it is unnecessary. The compiler knows if two methods have the same declaration, and will issue an error if they do. The compiler also knows how to find the proper method declaration, or whether one exists at all, by how the method is called. Finally, if an overloaded method has one or more declared versions that aren't actually called, the compiler knows this and will not emit the method during compilation.

Class Methods

Class methods are special types of methods that are sometimes referred to as "static" methods. They are callable from class instances and also directly from class references where no instance of the class exists, and are useful for implementing functions and procedures that should be encapsulated within the context of a class, but don't need to access class instance variables, properties, or methods. Class methods can, however, access any class variables or class properties that are also declared in the same class. Please see the Variables and Properties topics for more information on class variables and properties.

Class methods are declared by prefacing a method declaration with the **class** keyword. For example, the following class declaration includes a class variable that keeps track of how many instances of the class have been created along with a class method that returns the creation count:

```
TMyClass = class
   private
      class FCreateCount: Integer;
   public
      constructor Create; override;
      class function GetCreateCount: Integer;
   end;

implementation

constructor TMyClass.Create;
begin
   inherited Create;
   Inc(FCreateCount);
end;
```

The GetCreateCount class method above could be accessed in two different ways. The first way is by referring to the GetCreateCount class method for an instance of the TMyClass class:

```
procedure ShowCreateCount;
var
   TempInstance: TMyClass;
begin
   TempInstance:=TMyClass.Create;
   try
      window.alert(IntToStr(TempInstance.GetCreateCount));  // window class
      is in WebDOM unit
   finally
      TempInstance.Free;
   end;
end;
```

The second way is by using a direct class reference:

```
procedure ShowCreateCount;
begin
   window.alert(IntToStr(TMyClass.GetCreateCount));
end;
```

The second way is easiest when you don't have an instance of the class available.

Class methods are very useful for implementing factory classes that create instances, implementing singleton instances of classes, and implementing namespaces for code that otherwise would use normal functions and procedures declared outside of a class.

Constructors and Destructors

Constructors and destructors are special methods that handle the process of creating a class instance and destroying it. Because Elevate Web Builder compiles into JavaScript, these methods aren't explicilty allocating and deallocating memory. However, they are still crucial to ensuring that resources are properly allocated and initialized during the creation of class instances, and that resources are properly disposed of during the destruction of class instances. The base TObject class declaration contains both a constructor called **Create** and a destructor called **Destroy**.

Note
Constructors and destructors are completely optional. If a class declaration doesn't contain a constructor and/or destructor, then the ancestor class's constructor and/or destructor is used instead. If the ancestor class doesn't contain a constructor and/or destructor, then it's ancestor class is used and so on, until the base TObject class is reached by the compiler.

Constructors

Constructors are declared by prefacing a method declaration with the keyword **constructor**. Constructors must be declared as a procedure called **Create** with no result type declaration due to the fact the result is implicitly an instance of the class in which the declaration exists. Trying to declare a constructor with a different name or with a result type will cause a compiler error. If the declared constructor does not accept any parameters, then it must also be declared as an **override** of the base TObject Create constructor. Constructors can be overloaded, so it is possible to declare different constructors with different parameters. Finally, all constructors must be declared in the **public** scope of the class declaration and cannot be declared in any other scope.

Warning
Do not call constructors on instances of classes. Constructors are class, or static, methods, and should only be called on a class type itself in order to create an instance of the class.

The following is an example of a class that declares both an override of the base TObject Create constructor, as well as creates its own overloaded constructor:

```
interface

   TCustomer = class
      private
         FID: Integer;
         FName: String;
      public
         property ID: Integer read FID write FID;
         property Name: String read FName write FName;
      end;

   TCustomers = class
      private
```

```
         FCustomers: TObjectList;  // TObjectList class is declared
                                   // in the WebCore unit
         procedure CreateDemoCustomers(Value: Integer);
         function GetNumCustomers: Integer;
         function GetCustomer(ID: Integer): TCustomer;
         function GetCustomer(const Name: String): TCustomer;
      public
         constructor Create; override;
         constructor Create(NumDemoCustomers: Integer);
         property NumCustomers: Integer read GetNumCustomers;
         property Customer[ID: Integer]: TCustomer read GetCustomer; default;
         property Customer[const Name: String]: TCustomer read GetCustomer;
      default;
      end;

implementation

constructor TCustomers.Create;
begin
   inherited Create;
   FCustomers:=TObjectList.Create;
end;

constructor TCustomers.Create(NumDemoCustomers: Integer);
begin
   Create;
   CreateCustomers(NumDemoCustomers);
end;

procedure TCustomers.CreateDemoCustomers(Value: Integer);
var
   I: Integer;
   TempCustomer: TCustomer;
begin
   for I:=0 to Value-1 do
      begin
      TempCustomer:=TCustomer.Create;
      with TempCustomer do
         begin
         ID:=I;
         Name:='Demo Customer #'+IntToStr(I);
         end;
      FCustomers.Add(TempCustomer);
      end;
end;

function TCustomers.GetNumCustomers: Integer;
begin
   Result:=FCustomers.Count;
end;

function TCustomers.GetCustomer(ID: Integer): TCustomer;
var
   I: Integer;
begin
   Result:=nil;
   for I:=0 to FCustomers.Count-1 do
      begin
      if (TCustomer(FCustomers[I]).ID=ID) then
         begin
         Result:=TCustomer(FCustomers[I]);
         Break;
         end;
      end;
```

```
end;

function TCustomers.GetCustomer(const Name: String): TCustomer;
var
   I: Integer;
begin
   Result:=nil;
   for I:=0 to FCustomers.Count-1 do
      begin
      if SameStr(TCustomer(FCustomers[I]).Name,Name) then
         begin
         Result:=TCustomer(FCustomers[I]);
         Break;
         end;
      end;
end;
```

Note
 The above class declaration includes default array properties. Please see the Properties topic for more information on default array properties.

Destructors

A destructor is declared by prefacing a method declaration with the keyword **destructor**. There can be only one destructor per class and it must be declared as a procedure called **Destroy** that overrides the base TObject Destroy destructor and has no parameters. Trying to declare a destructor with a different name or with parameters will cause a compiler error. Finally, the destructor must be declared in the **public** scope of the class declaration and cannot be declared in any other scope.

The above example for constructors is an example of a class with an overridden Destroy destructor.

Free Method

Do not call the Destroy method above directly, use the TObject Free method instead. The Free method performs an extra crucial step that the Destroy method does not: the Free method checks to see if the calling instance variable is already nil, and only calls the Destroy method if the instance variable is not nil.

Note
 The only exception to the above rule is when calling the inherited Destroy method from within an ancestor class's Destroy method. That is a perfectly valid way to call the Destroy method directly.

Self

Use the special Self keyword in order to reference the current class instance from within a method. This is useful for situations where local variable or parameter names may conflict with a specific variable, method, or property name of the class in which the method resides, and so those identifiers need to be prefixed with the Self keyword.

8.17 Properties

Properties are one of the fundamental ways that Object Pascal provides encapsulation in classes. You can have properties directly reference variables or you can also use methods to control the reading and/or writing of variables. This helps to further hide the implementation details of a class and provide an easy-to-use class interface.

A property is declared as follows:

```
property <Property Name>: <Type Name> read <Variable Name>|<Method Name>
                                     [write <Variable Name>|<Method Name>]
                                     [default <Default Expression>]
                                     [description <Description>]
                                     [;default];
```

All properties must specify a variable or method name in the **read** clause, but the **write** clause is optional. If the write clause is not specified, then the property is implicitly read-only and cannot be assigned a value.

The following example shows a simple class with two read/write property declarations that directly reference variables:

```
interface

   TCustomer = class
      private
         FID: Integer;
         FName: String;
      public
         property ID: Integer read FID write FID;
         property Name: String read FName write FName;
      end;
```

The following example shows the same class, but modified to track modifications to any of the variables:

```
interface

   TCustomer = class
      private
         FID: Integer;
         FName: String;
         FModified: Boolean;
         procedure SetID(Value: Integer);
         procedure SetName(const Value: String);
      public
         property ID: Integer read FID write SetID;
         property Name: String read FName write SetName;
         property Modified: Boolean read FModified;
      end;

implementation
```

```
procedure TCustomer.SetID(Value: Integer);
begin
   if (Value <> FID) then
      begin
      FID:=Value;
      FModified:=True;
      end;
end;

procedure TCustomer.SetName(const Value: String);
begin
   if (not SameStr(Value,FID)) then
      begin
      FName:=Value;
      FModified:=True;
      end;
end;
```

The default clause specifies the default value for a property, and is optional. This default value is used to determine if the property should be streamed or not. If a default value is not provided for a property, and the property is published, then it will always be streamed when an instance of the owner class is streamed.

The description clause specifies the description for a property, and is also optional. This description appears in the Elevate Web Builder IDE's Object Inspector when the property is declared in the published scope, or is promoted to the published scope in an ancestor class.

The terminating default clause is different from the default value clause above. It is used to specify default array properties and default event properties. See below for more information on default array properties. A default event property is used by the Elevate Web Builer IDE to determine which event handler should be created when a developer double-clicks on a control in the form designer.

Array Properties

There is one special type of property that is a fairly powerful construct, and that is the array property. An array property is declared as follows:

```
property <Property Name>[[const] <Parameter Name>: <Type Name>]: <Type Name>
              read <Array Variable Name>|<Method Name>
              [write <Array Variable Name>|<Method Name>]; [default;]
```

An array property acts like an array but does not necessarily use an array variable for the read and/or write clauses of the property. Such a property can use methods instead of array variables, with the read clause requiring a method that accepts an identical single parameter that matches the array property parameter declaration and a write clause requiring a method that accepts the same parameter name and type and an additional parameter that can have any name that you wish, but whose type must match the type of the property.

Note

If you specify an array variable in the read or write clause, then the array property parameter must be an Integer type to reflect the ordinal index into the array.

For example, the following read-only property declares an array property that uses a method to return a specific class instance from an internal list:

```
TCustomers = class
   private
      FCustomers: TObjectList;
      function GetCustomer(ID: Integer): TCustomer;
   public
      property Customer[ID: Integer]: TCustomer read GetCustomer;
   end;
```

You would reference the Customer array property as follows:

```
var
   TempCustomers: TCustomers;
begin
   TempCustomers:=TCustomers.Create(10);
   try
      window.alert(TempCustomers.Customer[2].Name);
   finally
      TempCustomers.Free;
   end;
end;
```

Array properties make it very easy to hide the internal implementation of lists.

Default Array Properties

The above array property example illustrates a common problem with array properties used with classes that encapsulate lists: they tend to bloat the code by requiring the name of the array property to be specified. This is where default array properties are very useful. A default array property has the same declaration as a normal array property, but includes the **default** keyword after the declaration. To continue with the above example, let's change it to a default array property:

```
TCustomers = class
   private
      FCustomers: TObjectList;
      function GetCustomer(ID: Integer): TCustomer;
   public
      property Customer[ID: Integer]: TCustomer read GetCustomer; default;
   end;
```

You would reference the Customer default array property as follows:

```
var
   TempCustomers: TCustomers;
begin
   TempCustomers:=TCustomers.Create(10);
   try
```

```
      window.alert(TempCustomers[2].Name);
   finally
      TempCustomers.Free;
   end;
end;
```

Notice that since the Customer property is marked as the default array property, you no longer need to specify its name, only the instance variable name of the containing Customers class.

Overloaded Array Properties

One final interesting feature of array properties is that they can be overloaded so that you can have multiple such properties with the same name, but with different declarations. This is especially useful when you want to allow access to a list via different search types. For example, here is the above example with the Customer default array property overloaded with an additional declaration that uses a name parameter for retrieving a customer instead of an ID:

```
TCustomers = class
   private
      FCustomers: TObjectList;
      function GetCustomer(ID: Integer): TCustomer;
      function GetCustomer(const Name: String): TCustomer;
   public
      property Customer[ID: Integer]: TCustomer read GetCustomer; default;
      property Customer[const Name: String]: TCustomer read GetCustomer;
      default;
   end;
```

You could then reference the Customer default array property in both ways:

```
var
   TempCustomers: TCustomers;
begin
   TempCustomers:=TCustomers.Create(10);
   try
      window.alert(TempCustomers[2].Name);
      window.alert(IntToStr(TempCustomers['Demo Customer #2'].ID));
   finally
      TempCustomers.Free;
   end;
end;
```

Class Properties

Class properties are special types of properties that are sometimes referred to as "static" properties. They can be referenced from class instances and also directly from class references where no instance of the class exists, and are useful for implementing properties that should be encapsulated within the context of a class, but don't need to access class instance variables, properties, or methods. Class properties can, however, access any class variables or class methods that are also declared in the same class. Please see the Variables and Methods topics for more information on class variables and methods.

Class properties are declared by prefacing a property declaration with the **class** keyword. For example, the following class declaration includes a class variable that keeps track of how many instances of the class have been created along with a class property that returns the creation count:

```
TMyClass = class
   private
      class FCreateCount: Integer;
   public
      constructor Create; override;
      class property CreateCount: Integer read FCreateCount;
   end;

implementation

constructor TMyClass.Create;
begin
   inherited Create;
   Inc(FCreateCount);
end;
```

The CreateCount class property above could be accessed in two different ways. The first way is by referring to the CreateCount class property for an instance of the TMyClass class:

```
procedure ShowCreateCount;
var
   TempInstance: TMyClass;
begin
   TempInstance:=TMyClass.Create;
   try
      window.alert(IntToStr(TempInstance.CreateCount));  // window class is
      in WebDOM unit
   finally
      TempInstance.Free;
   end;
end;
```

The second way is by using a direct class reference:

```
procedure ShowCreateCount;
begin
   window.alert(IntToStr(TMyClass.CreateCount));
end;
```

The second way is easiest when you don't have an instance of the class available.

8.18 Events

Events are declared just like properties in classes, but read/write a method reference instead of a string, integer, etc. value. A method reference is similar to a class instance reference, but instead of referring to a class instance it refers to a method. Method references are useful because they can be called just like a normal method, but can also be assigned to variables and passed as parameters to other methods, procedures, or functions. This allows you to assign specific **behaviors** to the event properties of a class, swap such behaviors in and out with other behaviors, or assign no behavior at all.

Before an event can be declared, the event's method reference type must be declared. A method reference type is declared as follows:

```
<Type Name> = function/procedure ([<Parameters>])[: Type Name>] of object;
```

The procedure or function declaration is identical to a normal procedure or function prototype, except that a procedure or function name is not specified.

For example, consider the following method reference type and class/event declarations:

```
interface

   TStartEvent = procedure (StartingVehicle: TVehicle) of object;

   TVehicle = class
      private
         FOnStart: TStartEvent;
      public
         property OnStart: TStartEvent read FOnStart write FOnStart;
         procedure Start;
      end;

implementation

procedure TVehicle.Start;
begin
   if Assigned(FOnStart) then
      FOnStart(Self);
end;
```

Defining Event Handlers

Once an event is declared as a property in a class, it is still not very useful until the event is assigned an actual method reference. This type of method reference is referred to as an event handler. For example, the following code creates an instance of the TVehicle class and assigns an OnStart event handler:

```
interface

   TGarage = class
      private
```

```
         FVehicle: TVehicle;
      protected
         procedure DoVehicleStart(StartingVehicle: TVehicle);
      public
         constructor Create; override;
         destructor Destroy; override;
      end;

implementation

constructor TGarage.Create;
begin
   inherited Create;
   FVehicle:=TVehicle.Create;
   FVehicle.OnStart:=DoVehicleStart;
   FVehicle.Start;
end;

destructor TGarage.Destroy;
begin
   FVehicle.Free;
   inherited Destroy;
end;

procedure TGarage.DoVehicleStart(StartingVehicle: TVehicle);
begin
   window.alert('Vehicle has been started');
end;
```

Note
Event handlers are always called from the context of the class that contains the event handler. So, in the above example when the DoVehicleStart event handler is called, it will be called from the context of a TGarage class instance.

Method reference assignments and parameters must be type-compatible with the declared type of the target variable or parameter. To illustrate this concept, suppose that the above DoVehicleStart event handler was declared as follows:

```
TGarage = class
   private
      FVehicle: TVehicle;
   protected
      procedure DoVehicleStart;
   public
      constructor Create; override;
      destructor Destroy; override;
   end;
```

This would result in a compiler error on the source line where the event handler is assigned:

```
constructor TGarage.Create;
begin
   inherited Create;
   FVehicle:=TVehicle.Create;
   FVehicle.OnStart:=DoVehicleStart;  // Compiler error here !!!
   FVehicle.Start;
end;
```

The reason for the compiler error is simple: the TStartEvent type of the OnStart event is declared with a TVehicle parameter, and the DoVehicleStart method is not declared with any parameters.

Clearing Event Handlers

Just as you can attach a behavior to the event property of a class in the form of an event handler, you can also remove that behavior by assigning nil to the event property:

```
constructor TGarage.Create;
begin
   inherited Create;
   FVehicle:=TVehicle.Create;
   FVehicle.OnStart:=DoVehicleStart;
   FVehicle.Start;
   FVehicle.OnStart:=nil; // Clear event handler
end;
```

Warning
 Because the method reference variables that are used with event properties can be nil, you should always check for this before trying to call such method reference variables. The best way to do so is by using the Assigned function, as illustrated in the TVehicle.Start method implementation:

```
procedure TVehicle.Start;
begin
   if Assigned(FOnStart) then
      FOnStart(Self);
end;
```

Default Events

Components and controls that will be used in the component library can have a default event property. A default event property is the event handler that will be created if the user double-clicks on a component or control in the Form Designer. For example, the following TVehicle class has been slightly modified from the above example. It is now a descendant of the TComponent class so that it can be installed into the component library, and now defines the OnStart event as the default event in the published section of the class:

```
interface

   TStartEvent = procedure (StartingVehicle: TVehicle) of object;

   TVehicle = class(TComponent)
```

```
      private
         FOnStart: TStartEvent;
      public
         procedure Start;
      published
         property OnStart: TStartEvent read FOnStart write FOnStart; default;
      end;

implementation

procedure TVehicle.Start;
begin
   if Assigned(FOnStart) then
      FOnStart(Self);
end;
```

A default event is only used in the form designer if the event property is published. Please see the Scope topic for more information on published properties.

8.19 Scope

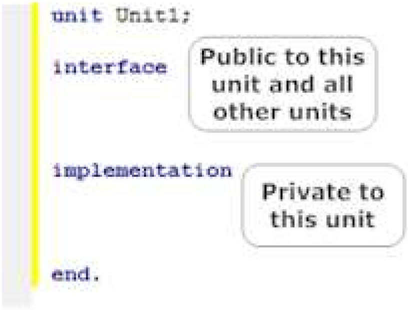
The scope (visibility) of a constant, type/class, or function/procedure declaration is determined by where it appears in a unit, class, or function/procedure.

Unit Scope

The scope of declarations in a unit are determined by whether they are declared in the interface or implementation section of a unit:

- Any declaration in the interface section of a unit is visible to all other declarations or function/procedure implementations in either the interface or implementation sections of the same unit, as well as being visible to the same sections in any units that reference the unit. The interface section of a unit is essentially "public" to everything.

- Any declaration in the implementation section of a unit is visible to all other declarations in the same implementation section only.



```
unit Unit1;

interface

implementation

end.
```

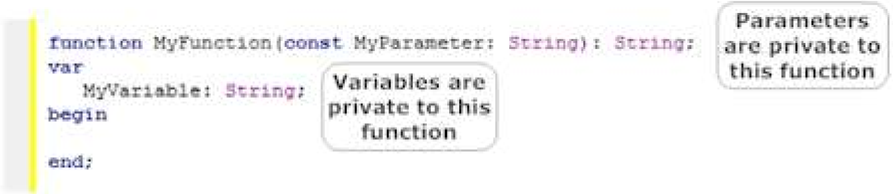
Unit Interface/Implementation Scope

The visibility of type and class declarations is determined in top-to-bottom fashion within an interface or implementation section of a unit. For example, if the TClassA class descends from the TClassB class, but TClassB is declared before TClassA in the same unit section, the compiler will issue an error . This is because the compiler cannot "see" the TClassA declaration at the point of the TClassB declaration:

```
interface

type

   TClassB = class(TClassA) // Compiler error here, TClassA class
                            // doesn't exist yet
      end;

   TClassA = class
      private
         FMemberVariable: String;
      public
         property MemberProperty: String read FMemberVariable;
      end;
```

Function/Procedure Implementation Scope

The implementations of functions and procedures only have one level of scope, and that is the scope of any parameters or local variables, as well as the special Result variable for functions. Parameters and local variables can only be referenced within the function/procedure in which they are declared.



```
function MyFunction(const MyParameter: String): String;
var
   MyVariable: String;
begin

end;
```

Parameters are private to this function
Variables are private to this function

Function/Procedure Scope

Class methods (functions or procedures declared as part of a class) add an additional level of scope that is evaluated after any local variables or parameters. Class methods can access any member variables, properties, or methods that are declared anywhere within the same class in which the referencing method is declared. Class methods can also access any public or protected member variables, properties, or methods that are declared within any ancestor classes of the class in which the referencing method is declared.

Note
In order to specifically reference the current instance of a class from within a class method, use the special **Self** keyword.

See below for more information about the various levels of scope (private, protected, public, published) within a class declaration:

```
interface

type

   TClassA = class
      private
         FMemberVariable: String;
      protected
         procedure DoSomethingMethod;
      public
         property MemberProperty: String read FMemberVariable;
      end;

   TClassB = class(TClassA)
      public
         procedure DoSomethingElseMethod;
      end;

implementation

{ TClassA Implementation }
```

```
procedure TClassA.DoSomethingMethod;
begin
   FMemberVariable := 'Test';  // Can access this variable because it is also
                               // declared within the TClassA declaration
end;

{ TClassB Implementation }

procedure TClassB.DoSomethingElseMethod;
begin
   DoSomethingMethod;  // Can access this protected method because it is
                       // declared within the ancestor TClassA declaration
end;
```

There is one exception to the normal scoping rules of a function/procedure, and that is the with statement. The with statement can introduce an object instance as a new level of scope that overrides the normal scope of any local variables, parameters, or class variables/properties/methods.

Class Scope

A class declaration can have up to four different levels of scope for any member variables, properties, or methods:

Scope	Description
Private	Any member variables, properties, or methods declared in this scope are only visible to the properties or methods of the same class declaration.
Protected	Any member variables, properties, or methods declared in this scope are only visible to the properties or methods of the same class declaration, or any descendant class declarations.
Public	Any member variables, properties, or methods declared in this scope are visible everywhere, subject to the scoping rules of the referencing declaration or code block.
Published	Same as Public, but in addition, all properties declared in this scope are streamable and will appear in the Elevate Web Builder IDE's Object Inspector.

Naming Conflicts and Scope

In certain cases, the scoping rules are used by the compiler to resolve naming conflicts. For example, if the implementation of a method uses a local variable that uses the same name as a member variable of the class in which the method is declared, the local variable scope will take precedence:

```
interface

type

   TMyClass = class
      private
         MyVariable: String;
         procedure MyMethod;
      end;

implementation

procedure TMyClass.MyMethod;
var
   MyVariable: String; // This declaration overrides the scope of the class
begin
   MyVariable := 'Test';       // This will be assigned to the local
                               // MyVariable variable
   Self.MyVariable := 'Test';  // This will be assigned to the MyVariable
      member
                               // variable of the TMyClass class
end;
```

Certain naming conflicts are impossible because the compiler will not permit them. For example, you cannot give a local variable the same name as a parameter in the same function/procedure declaration, nor can you name a local variable the special "Result" variable name used for returning results from functions.

8.20 Casting Types

Casting is the process of converting a target value of one type to another type in order to use the value in a different type context. This is accomplished by enclosing the target value with the target type name and parentheses:

```
<Type Name>(<Target Value>)
```

There are type compatibility rules that determine whether a particular cast operation is valid, and invalid cast attempts will result in a compiler error. The following table details the various types and their valid target cast types:

Source Type	Valid Target Types
Integer	Integer Boolean Double DateTime Enumeration
Double	Double
String	String Char
Char	Char String
DateTime	DateTime Integer
Boolean	Boolean Integer
Enumeration	Enumeration Integer
Class Instance Class Type	Any same class type or ancestor class type

Casting is particularly useful for functions or procedures that accept a parameter of a base class type, but need to act on the various descendant class types in specific ways. For example, the following code shows how one would use the **is** operator to determine if the parameter is of the TComponent type and, if so, displays its name:

```
procedure DisplayName(Value: TObject);
begin
   if (Value is TComponent) then
      window.alert(TComponent(Value).Name);
end;
```

8.21 Exception Handling

Exceptions are special classes in Elevate Web Builder that represent an error that has been raised by code in the application, or by the web browser itself. All exceptions descend from the base Exception class defined in the WebCore unit. At design-time, exceptions are handled by the execution engine in the IDE. At run-time, the Exception class is mapped to the base Error class present in the standard JavaScript run-time.

> **Warning**
> While you can create new exception classes that descend from the base Exception class for design-time usage, you cannot do so for run-time usage. Some of the modern web browser implementations of JavaScript do not properly deal with exception class descendants in terms of reporting the proper error message and source line number of the error.

Raising Exceptions

You can raise an exception at any time by using the **raise** statement. The raise statement requires an Exception class (or descendant) instance as its only argument:

```
raise <Exception Class Instance>;
```

Once an exception is raised, execution stops immediately and the process of unwinding the call stack and triggering exception handlers begins.

The following example shows how to raise an exception that indicates that a parameter was not assigned a valid positive value:

```
function AddValues(A,B: Integer): Integer;
begin
   if (A < 0) then
      raise Exception.Create('First parameter '+IntToStr(A)+' cannot be
      negative');
   if (B < 0) then
      raise Exception.Create('Second parameter '+IntToStr(A)+' cannot be
      negative');
   Result:=(A+B);
end;
```

Handling Exceptions

Once an exception has been raised, either by the design-time or run-time execution environment, or the application code, execution immediately stops and the call stack is unwound, with any exception-handling blocks executed as necessary during this process.

Exceptions can be handled by using a **try..except** code block. The syntax of a try..except code block is:

```
try
   <Statements>
except
   <Exception-handling statements>
end;
```

A try..except code block catches any exceptions that occur with the try and except keywords, preventing them from escaping the current function or procedure and unwinding the call stack.

You can access the current exception from inside the except portion of the try..except code block by using the **on** statement, which uses the following syntax:

```
on <ExceptionInstanceVariable>: <ExceptionClass> do
   <Statements>

on <ExceptionClass> do
   <Statements>
```

There are two different variations of the on statement:

- The first variation specifies a local variable name followed by a colon and an exception class name. This is the most useful type of on statement because it allows you to capture the existing exception in a local variable. This is important when you want to examine the error Message or log it for later examination. The exception class name is used to filter which exception classes are handled by the on statement. However, you should always use the Exception class here due to the fact that the base Exception class is the only recommended exception class to use (see above).

- The second variation specifies an exception class name only. This variation is not particularly useful due to the fact that the base Exception class is the only recommended exception class to use (see above).

The following is an example of using the on statement to log an error message to the Messages panel in the IDE using the LogOutput method:

```
begin
  try
     // Statements that raise exception
  except
     on E: Exception do
        LogOutput(E.Message);
  end;
end;
```

Please see the Debugging topic for more information on the LogOutput method.

Re-Raising Exceptions

You can re-raise an existing exception by using the **raise** statement without any arguments. Having the ability to re-raise exceptions is useful in situations where you want to do something with an exception, such as log its associated message, before allowing the call stack to proceed unwinding.

Note
Re-raising exceptions can only be done from within the except portion of a try..except code block, and an attempt to do so outside of this context will cause a compiler error.

The following example expands upon the above example by also re-raising the exception after logging the error message:

```
begin
  try
     // Statements that raise exception
  except
     on E: Exception do
        LogOutput(E.Message);
     raise;
  end;
end;
```

Ensuring Code Execution After Exceptions

It is often necessary to ensure that certain statements execute, regardless of whether an exception is raised or not. This is accomplished by using a **try..finally** code block. The syntax of a try..finally code block is:

```
try
   <Statements>
finally
   <Statements>
end;
```

Any statements specified within the finally portion of the try..finally code block will always be executed, which is useful for situations where class instances, or other types of resources, have been allocated and need to be disposed of.

The following example shows the method of a class that toggles an internal Boolean variable in the class, and must ensure that the variable is toggled again before the method exits:

```
procedure TMyClass.Execute;
begin
   FExecuting:=True;
   try
      // Executing
   finally
      FExecuting:=False;
   end;
end;
```

Note
A try..finally code block also applies to the **exit** statement. If an exit statement is specified inside of a try..finally code block, the finally portion of the code block will be executed before the function or procedure actually exits.

Visual Application Exceptions

If an exception is not handled at run-time with a try..except code block in a visual application, the exception will result in an Elevate Web Builder message dialog appearing with the error message and source line number. If you do not want this to occur, you can define a TApplication OnError event handler for the global Application instance that is automatically created for visual applications. Returning True from this event handler will indicate to the web browser that the error was handled and will prevent the browser from displaying an error dialog.

8.22 External Interfaces

Sometimes it is necessary to make calls from an application to external code such as 3rd party JavaScript code or the built-in classes available as part of the web browser's DOM (Document Object Model) class hierarchy. The DOM is the core framework in a modern web browser that allows any JavaScript code to create and manipulate elements in an HTML or XML document, as well as parts of the web browser itself. Elevate Web Builder includes a fairly complete external interface to the DOM in the WebDOM unit that is part of the runtime code included with the product.

The Elevate Web Builder compiler requires that an external interface be declared for any external DOM classes or JavaScript code before such classes or code can be used in an application. External interfaces are only interfaces and do not include any type of implementation. Using external interfaces will ensure that the benefits of compile-time type checking are applied to external code as well as the Object Pascal code in the application, thus allowing for more reliable applications.

Please see the Modifying Project Options and Using the Project Manager topics for more information on including external JavaScript source files with an application.

External Declarations

You **must** include the WebDOM unit in the uses clause of the interface section of the unit in which any external class declarations are included. The WebDOM unit is necessary for obtaining certain base class declarations.

Any constant, variable, type, class, procedure, or function can be declared as **external**. For example, the DOM in the web browser includes a global variable called "window" that is an instance of the Window DOM class. Elevate Web Builder represents both in the WebDOM unit as follows:

```
interface

   external TWindow emit Window = class
      public
         { Properties }
         property closed: Boolean read;
         property defaultStatus: String read write;
         property document: TDocument read;
         property event: TEvent read; // IE-only
         property frames: TWindowList read;
         property history: THistory read;
         property innerHeight: Integer read; // Supported by IE9 or higher
         property innerWidth: Integer read; // Supported by IE9 or higher
         property localStorage: TStorage read;
         property location: TLocation read;
         property name: String read write;
         property navigator: TNavigator read;
         property opener: TWindow read;
         property orientation: Integer read; // Mobile platforms only
         property outerHeight: Integer read; // Not supported by IE
         property outerWidth: Integer read; // Not supported by IE
         property pageXOffset: Integer read; // Not supported by IE
         property pageYOffset: Integer read; // Not supported by IE
         property parent: TWindow read;
         property screen: TScreen read;
         property screenLeft: Integer read; // IE-only
         property screenTop: Integer read; // IE-only
         property screenX: Integer read; // Not supported by IE
         property screenY: Integer read; // Not supported by IE
         property sessionStorage: TStorage read;
         property status: String read write;
         property top: TWindow read;
         property window: TWindow read;
         { Events }
         property onblur: TEventHandler read write;
         property onerror: TErrorEventHandler read write;
         property onfocus: TEventHandler read write;
         property onload: TEventHandler read write;
         property onresize: TEventHandler read write;
         property onunload: TEventHandler read write;
         { Methods }
         procedure addEventListener(const type: String; listener:
      TEventHandler;
                                    useCapture: Boolean);
         procedure alert(const message: String);
         procedure blur;
         procedure cancelAnimationFrame(animationId: Integer);
         procedure clearInterval(intervalId: Integer);
         procedure clearTimeout(timeoutId: Integer);
         procedure close;
         function confirm(const question: String): Boolean;
         procedure detachEvent(const type: String; handler: TEventHandler);
         procedure focus;
         function getComputedStyle(elt: TDOMElement; const pseudoElt:
      String): TCSS2Properties;
         procedure moveBy(dx, dy: Integer);
         procedure moveTo(x, y: Integer);
         function open(const url: String; const name: String=''; const
      features: String='';
                       replace: Boolean=False): TWindow;
         procedure print;
         function prompt(const message: String; default: String): String;
         procedure removeEventListener(const type: String; listener:
      TEventHandler;
                                       useCapture: Boolean);
         function requestAnimationFrame(callback: TAnimationHandler): Integer;

         procedure resizeBy(dw, dh: Integer);
         procedure resizeTo(w, h: Integer);
         procedure scrollBy(dx, dy: Integer);
         procedure scrollTo(x, y: Integer);
         function setInterval(code: TIntervalHandler; intervalId: Integer):
      Integer;
         function setTimeout(code: TIntervalHandler; intervalId: Integer):
      Integer;
      end;

var
   external window: TWindow;
```

> **Warning**
> Because JavaScript is case-sensitive, all external declarations are **case-sensitive** and must match the required case of the external Javascript declarations for the same entities. This only applies to the external declarations. All Object Pascal code that calls the external code can still use any case desired, and the compiler will automatically make sure that the proper case is used in the emitted code for the application.

The rules and exceptions for external declarations are:

- External class declarations cannot contain private or protected members, only public members.
- External class instances do not necessarily follow the same instantiation rules regarding member variables. Non-external classes are guaranteed to have all of their member variables initialized to appropriate values, but this is not necessarily true for external classes.
- External classes can still be created with the Create method and freed with the Free method, but internally the compiler will emit slightly different code than it does for non-external classes.
- External classes can only inherit from other external classes, and non-external classes can only inherit from non-external classes.
- The read and write clauses for properties in external class declarations do not refer to any member variables or methods.
- You can use the **emit** clause to control the class name/namespace used when the compiler emits references to the class when creating new instances. This is useful when instantiating JavaScript objects that are nested within namespaces. For example, Google Maps integration requires the following emit clause:

```
external TGoogleMapOptions emit google.maps.MapOptions = class
```

8.23 Debugging

There are currently two ways to debug Elevate Web Builder applications at runtime in a web browser: by using a runtime function to log debug messages to the Messages panel in the IDE, and by using the built-in debugging facilities in the web browser of your choosing. You can use the debug message option with the embedded web browser in the IDE, but not the debugging facilities.

LogOutput Procedure

For simple debugging needs, make sure that the internal web server is the selected web server in the IDE and that the internal web server is running (see Running a Project). Then, include the WebHTTP unit in the uses clause of the unit that you wish to debug. Finally, call the LogOutput procedure where necessary, passing any debug messages to the procedure as a single String parameter:

```
procedure LogOutput(const Msg: String;
                    const LogURL: String=DEFAULT_LOG_URL);
```

By default, the LogOutput procedure will send all debug output to the internal web server by using the following URL:

```
http://localhost/log
```

Any messages passed to the LogOutput method will automatically appear in the Messages panel in the IDE.

Note
This also applies to applications that are run in an external browser session. As long as the application is accessed via a **localhost** URL and is being loaded from the internal web server running in the IDE, all debug output will get routed to the Messages panel in the IDE.

If you are running the application in an external browser session on a completely different machine or device, then be sure to include the second parameter to the LogOutput procedure. This parameter should include the IP address/host name of the machine running the Elevate Web Builder IDE. For example, if you were running the application in a Chrome browser on an Android tablet, and the Elevate Web Builder IDE was running on the same LAN at IP address 192.168.0.2, then you would use the following value for the second LogOutput parameter:

```
http://192.168.0.2/log
```

Web Browser Debuggers

For more complex debugging needs, make sure that the **Compress Output** option is **not** checked on the

Compilation page of the project options for your project, compile the project, and then run the application in an external web browser session with the web browser's debugger enabled. With Internet Explorer, FireFox, and Chrome, you can access the debugger by using the F12 key to open the developer tools panel while in the browser. The one major downside to this type of debugging is that you must debug the emitted JavaScript code, and not the Object Pascal code. Fortunately, though, the two are very similar in content and layout, and so the emitted JavaScript code is usually fairly easy to debug.

Note
In general, it should not be necessary to debug your code using the web browser debugging facilities. If you find yourself doing so often, please let us know and we'll use this information to help better plan our future implementation of a built-in debugger.

8.24 Asynchronous Calls

Elevate Web Builder supports asynchronous procedure/function calls using the special **async** keyword. Asynchronous calls allow the developer to queue a procedure/function call in the browser so that it is run as part of the message queue processing for the main UI thread in the browser. Asynchronous calls are available **only** at runtime, and will cause a component library compilation error if used in any design-time code.

How Asynchronous Calls Are Executed

Because asynchronous calls are added to the message queue for the main UI thread in the browser, they are executed in a first-in, first-out (FIFO) manner. This means that there may be a delay between when the asynchronous call is made and when the call is actually executed. Also, asynchronous calls are emitted by the compiler as Javascript **closures**. Closures are functions that are dynamically created and capture the entire run-time scope of their parent execution context. Whenever a closure is actually executed, it will do so using the same scope that was present when the closure was created. Closures are ideal for asynchronous calls, because they need to capture the state of all variables and parameters so that they are available when the call is actually executed.

Executing an Asynchronous Call

To make an asynchronous procedure/function call, simply preface the call with the async keyword. For example,

```
procedure TForm1.Button1Click(Sender: TObject);
begin
   async CreatePanel(0);
end;
```

will queue up a call to the CreatePanel procedure so that it will run in the next round of message processing in the browser. Because the compiler will emit a closure for this call, the value of any local variables or parameters will be properly captured, even if the parent method that is calling the function/procedure has finished executing.

Mixing Synchronous/Asynchronous Calls

Because the main UI thread in the browser is used for executing all code, any **synchronous** code will execute **before** any asynchronous calls that are queued in the message queue. This is important to understand because it determines how you should combine synchronous and asynchronous calls to achieve the desired outcome.

For example, suppose that you want to create a large number of panels in a container, and want to show a progress dialog while the panels are created. To do this, you would normally do something like this:

```
procedure TForm1.CreatePanels;
var
   I: Integer;
```

```
begin
   for I:=1 to 100 do
      TPanel.Create(Self);
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
   ShowProgress('Creating panels...');
   CreatePanels;
   HideProgress;
end;
```

However, if you were to execute the above code in the browser, you will see that the panels are created, but the progress dialog will never show. This is because the UI updates for the ShowProgress call will not be executed until any other currently-executing code has completed. In this case, this is the CreatePanels and HideProgress calls, so the ShowProgress UI updates will get merged with the HideProgress UI updates, and the progress dialog will never get shown (or will be shown/hidden so fast that you won't see it).

The key to fixing this problem is to allow the UI to update incrementally while we create the panels, and we'll use asynchronous calls to do so:

```
procedure TForm1.CreatePanel(I: Integer);
begin
   TPanel.Create(Self);
   Inc(I);
   if (I < 100) then
      async CreatePanel(I)
   else
      async HideProgress;
end;

procedure TForm1.Button1Click(Sender: TObject);
begin
   ShowProgress('Creating panels...');
   async CreatePanel(0);
end;
```

We **don't** want to use an asynchronous call to ShowProgress because we want it to be executed immediately so that it is the first UI update to occur. However, we do want to queue each CreatePanel call and the HideProgress call because doing so will force them to execute in-order after any UI updates from the ShowProgress call, as well as allow the UI to update during each panel creation.

Chapter 9
Function and Procedure Reference

9.1 Abs

Unit: Internal

```
function Abs(Value: Double): Double

function Abs(Value: Integer): Integer
```

The **Abs** function returns the absolute value of the input parameter. The return value is the same type as the input parameter.

Examples

```
X := Abs(-10);  // X is 10
X := Abs(100);  // X is 100
```

9.2 ArcCos

Unit: Internal

```
function ArcCos(Value: Double): Double

function ArcCos(Value: Integer): Double
```

The **ArcCos** function returns the arccosine, or inverse cosine, of the input parameter, which must be between -1 and 1. The return value is a Double value between 0 and Pi radians.

Examples

```
X := ArcCos(0.23290);  // X is 1.335737700525506
```

9.3 ArcSin

Unit: Internal

```
function ArcSin(Value: Double): Double

function ArcSin(Value: Integer): Double
```

The **ArcSin** function returns the arcsine of the input parameter, which must be between -1 and 1. The return value is a Double value between -Pi/2 and Pi/2 radians.

Examples

```
X := ArcSin(0.23290);  // X is 0.23505862626939056
```

9.4 ArcTan

Unit: Internal

```
function ArcTan(Value: Double): Double

function ArcTan(Value: Integer): Double
```

The **ArcTan** function returns the arc tangent of the input parameter. The return value is a Double value between -Pi/2 and Pi/2 radians.

Examples

```
X := ArcTan(0.23290);  // X is 0.22882093498523032
```

9.5 ArcTan2

Unit: Internal

```
function ArcTan2(Y, X: Double): Double

function ArcTan2(Y, X: Integer): Double
```

The **ArcTan2** function returns a value between -Pi and Pi radians that specifies the counter-clockwise angle between the positive X axis and the point represented by the X and Y input parameters.

Examples

```
X := ArcTan2(100,3000);  // X is 0.033320995878247196
```

9.6 Assigned

Unit: Internal

```
function Assigned(Value: TObject): Boolean

function Assigned(Value: function of object): Boolean

function Assigned(Value: procedure of object): Boolean

function Assigned(const Value: String): Boolean

function Assigned(const Value: array of <Type>): Boolean
```

Examples

The **Assigned** function returns whether the input parameter is nil or has been assigned a value. The input parameter must be an object instance, method reference (function or procedure of object), string, or array variable. The return value is a Boolean value.

Examples

```
var
   MyObject: TObject;
begin
   X := Assigned(MyObject);  // X is False
   MyObject := TObject.Create;
   X := Assigned(MyObject);  // X is True
end;

var
   MyString: String;
begin
   X := Assigned(MyString);  // X is False
   MyString := 'This is a test';
   X := Assigned(MyString);  // X is True
end;
```

9.7 BoolToStr

Unit: WebCore

```
function BoolToStr(Value: Boolean): String
```

The **BoolToStr** function returns 'True' if the input parameter is true, and 'False' if the input parameter is False. The return value is a String value.

Examples

```
X := BoolToStr(True);  // X is 'True'
```

9.8 Ceil

Unit: Internal

```
function Ceil(Value: Double): Integer

function Ceil(Value: Integer): Integer
```

The **Ceil** function returns the closest integer that is greater than or equal to the value of the input parameter. The return value is an Integer.

Examples

```
X := Ceil(-10.4);  // X is -10
X := Ceil(15.98);  // X is 16
```

9.9 Chr

Unit: Internal

```
function Chr(Value: Integer): Char
```

The **Chr** function returns the character representation of the Unicode code point input parameter. The return value is a Char.

Examples

```
X := Chr(220);  // X is 'Ü'
```

9.10 CompareStr

Unit: Internal

```
function CompareStr(const A, B: String): Integer
```

The **CompareStr** function compares the A input parameter string with the B input parameter string with case-sensitivity. The comparison is locale-insensitive. The return value is an Integer value of -1 if A is less than B, 0 if A is equal to B, and 1 if A is greater than B.

Examples

```
X := CompareStr('Absolute', 'Baseball');  // X is -1
```

9.11 CompareText

Unit: Internal

```
function CompareText(const A, B: String): Integer
```

The **CompareText** function compares the A input parameter string with the B input parameter string without case-sensitivity. The comparison is locale-insensitive. The return value is an Integer value of -1 if A is less than B, 0 if A is equal to B, and 1 if A is greater than B.

Examples

```
X := CompareText('Absolute', 'ABSOLUTE');  // X is 0
```

9.12 Copy

Unit: Internal

```
function Copy(const Value: String; Index: Integer;
              Count: Integer): String

function Copy(const Value: String; Index: Integer): String

function Copy(const Value: String): String

function Copy(const Value: array of <Type>; Index: Integer;
              Count: Integer): array of <Type>

function Copy(const Value: array of <Type>; Index: Integer): array of <Type>

function Copy(const Value: array of <Type>): array of <Type>
```

The **Copy** function returns a portion of the Value string or array input parameter. The optional Index input parameter specifies where to start the copy, and the optional Count input parameter specifies the length to copy. If the Count input parameter is not specified, then the copy will proceed until the end of the Value input parameter. If neither the Index or Count input parameters are specified, then an exact copy of the Value input parameter will be returned.

Note

Please remember that indexes into String values are 1-based, whereas indexes into arrays are 0-based. For more information on these types, please see the Types topic.

Examples

```
X := Copy('abcdef', 4, 3);  // X is 'def'
X := Copy('abcdef', 2);  // X is 'bcdef'
X := Copy([10,20,30,40], 0, 3);  // X is [10,20,30]
```

9.13 Cos

Unit: Internal

```
function Cos(Value: Double): Double

function Cos(Value: Integer): Double
```

The **Cos** function returns the cosine of the input parameter, which is an angle specified in radians. To convert an angle from degrees to radians, use the Radians function. The return value is a Double value between -1 and 1.

Examples

```
X := Cos(0.23290);  // X is 0.9730011668494914
```

9.14 CreateActiveXObject

Unit: Internal

```
function CreateActiveXObject: <External Object Instance>
```

The **CreateActiveXObject** function creates an external ActiveX object instance (Internet Explorer only) and returns it as the result.

> **Note**
> You must always cast the result of this function to the desired external class in order to be able to use the resulting instance in your code.

Examples

```
var
   TempDocument: TDocument;
begin
   // Must always cast the result to the desired class
   TempDocument:=TDocument(CreateActiveXObject('Microsoft.XMLDOM'));
   TempDocument.LoadXML(Value);
end;
```

9.15 CreateObject

Unit: Internal

```
function CreateObject(const ObjectLiteral: String): <External Object
      Instance>
```

The **CreateObject** function creates an external object instance from a JavaScript object literal and returns it as the result. This function can be especially useful with JS APIs that require that you create object instances using object literals.

Note
You must always cast the result of this function to the desired external class in order to be able to reference any properties or methods of the new external class instance from within Elevate Web Builder code.

Warning
This function internaly uses the JavaScript eval function in order to create the object. Be very careful about passing object literal strings that have been derived from an external source to this function.

Examples

```
type
   external TMyExternalObject emit MyJSAPI.MyExternalObject = class
      public
         property Name: String read write;
      end;

var
   TempObject: TMyExternalObject;
begin
   TempObject:=TMyExternalObject(CreateObject('{ name: ''My External Object''
      }'));
end;
```

9.16 Date

Unit: Internal

```
function Date: DateTime
```

The **Date** function returns the current date. The return value is a DateTime value.

Examples

```
X := DateToStr(Date);  // X is '2/13/2012'
```

9.17 DateTimeToStr

Unit: WebCore

```
function DateTimeToStr(Value: DateTime; UTC: Boolean=False): String
```

The **DateTimeToStr** function returns a formatted local or UTC date and time string for the DateTime input parameter. The format of the string is determined by the TFormatSettings ShortDateFormat and ShortTimeFormat properties. The return value is a String value.

Examples

```
A := StrToDateTime('2/13/2012 12:10 PM');
X := DateTimeToStr(A);  // X is '2/13/2012 12:10 PM'
```

9.18 DateTimeToISOStr

Unit: Internal

```
function DateTimeToISOStr(Value: DateTime): String
```

The **DateTimeToISOStr** function returns an ISO-8601-formatted date-time string value for the date-time input parameter. The return value is a String value.

Examples

```
X := DateTimeToISOStr(StrToDateTime('8/15/2015 12:40 PM'));  // X is
      '2015-08-15T16:40:00.000Z'
```

9.19 DateToStr

Unit: WebCore

```
function DateToStr(Value: DateTime; UTC: Boolean=False): String;
```

The **DateToStr** function returns a formatted local or UTC date string for the DateTime input parameter. The format of the string is determined by the TFormatSettings ShortDateFormat property. The return value is a String value.

Examples

```
A := StrToDateTime('2/13/2012');
X := DateToStr(A);  // X is '2/13/2012'
```

9.20 DayOf

Unit: Internal

```
function DayOf(Value: DateTime; UTC: Boolean=False): Integer
```

The **DayOf** function returns the day number of the input parameter in local or UTC time. The return value is an Integer value.

Examples

```
X := DayOf(Date);  // X is 13 (assuming a date of 02/13/2012)
```

9.21 Dec

Unit: Internal

```
procedure Dec(var Value: Integer)

procedure Dec(var Value: Integer; By: Integer)
```

The **Dec** procedure decrements the Integer input parameter by 1, or by the By input parameter, if specified.

Examples

```
X := 10;
Dec(X);
Y := IntToStr(X)  // Y is '9'
```

9.22 Degrees

Unit: Internal

```
function Degrees(Value: Double): Double

function Degrees(Value: Integer): Double
```

The **Degrees** function converts the input parameter, which is an angle specified in radians, to degrees. The return value is a Double value.

Examples

```
X := Degrees(92.398);  // X is 5294.01543544978
```

9.23 Delete

Unit: Internal

```
procedure Delete(const Value: array of <Type>; Index: Integer;
                 Count: Integer)

procedure Delete(const Value: array of <Type>; Index: Integer)
```

The **Delete** procedure deletes a portion of the Value array input parameter. The optional Index input parameter specifies where to start the deletion, and the optional Count input parameter specifies the number of array elements to delete. If the Count input parameter is not specified, then the deletion will proceed until the end of the Value input parameter.

Note
This procedure cannot be used with strings in Elevate Web Builder. Strings are immutable in JavaScript, and therefore cannot be modified in-place using procedures such as this. For more information on these types, please see the Types topic.

Examples

```
X := [10,20,30,40];
Delete(X, 2, 2); // X is [10,20] after the Delete call
```

9.24 DoubleToStr

Unit: Internal

```
function DoubleToStr(Value: Double; Decimals: Integer=-1): String
```

The **DoubleToStr** function returns a formatted string for the Double input parameter. The decimal separator used in the formatted string is **always** a period (.). The return value is a String value. You can use the optional Decimals parameter to specify that the return value is formatted to a specific number of decimal places.

Examples

```
A := StrToDouble('1200.548');
X := DoubleToStr(A);  // X is '1200.548'

A := StrToDouble('1200.548');
X := DoubleToStr(A, 1);  // X is '1200.5'
```

9.25 EncodeDate

Unit: Internal

```
function EncodeDate(Year: Integer; Month: Integer; Day: Integer;
                    UTC: Boolean=False): Integer
```

The **EncodeDate** function returns the local or UTC date from the year, month, and day input parameters. The return value is a DateTime value.

Examples

```
X := DateToStr(EncodeDate(2012,2,13));  // X is '2/13/2012'
```

9.26 EncodeDateTime

Unit: Internal

```
function EncodeDateTime(Year: Integer; Month: Integer; Day: Integer;
                        Hour: Integer; Minute: Integer; Second: Integer;
                        MSecond: Integer; UTC: Boolean=False): Integer
```

The **EncodeDateTime** function returns the local or UTC date and time from the year, month, day, hour, minute, second, and millisecond input parameters. The return value is a DateTime value.

Examples

```
X := DateTimeToStr(EncodeDateTime(2012,2,13,12,10,0,0)); // X is '2/13/2012
                                                                  // 12:10 PM'
```

9.27 EncodeTime

Unit: Internal

```
function EncodeTime(Hour: Integer; Minute: Integer; Second: Integer; MSecond:
      Integer;
                     UTC: Boolean=False): Integer
```

The **EncodeTime** function returns the local or UTC time from the hour, minute, second, and millisecond input parameters. The return value is a DateTime value.

Examples

```
X := TimeToStr(EncodeTime(12,10,0,0));  // X is '12:10 PM'
```

9.28 Exp

Unit: Internal

```
function Exp(Value: Double): Double

function Exp(Value: Integer): Double
```

The **Exp** function returns e raised to the power specified by the input parameter, where e is the base of the natural logarithm. The return value is a Double value.

Examples

```
X := Exp(0.523);  // X is 1.6870813093472114
```

9.29 FloatToStr

Unit: WebCore

```
function FloatToStr(Value: Double; Decimals: Integer=-1): String
```

The **FloatToStr** function returns a formatted string for the Double input parameter. The decimal separator used in the formatted string is determined by the TFormatSettings DecimalSeparator property. The optional Decimals input parameter determines the number of decimal places used in the formatted string. The return value is a String value.

Examples

```
A := StrToFloat('1200.548');
X := FloatToStr(A);  // X is '1200.548'

A := StrToFloat('1200.548');
X := FloatToStr(A,2);  // X is '1200.55'
```

9.30 Floor

Unit: Internal

```
function Floor(Value: Double): Integer

function Floor(Value: Integer): Integer
```

The **Floor** function returns the closest integer that is less than or equal to the value of the input parameter. The return value is an Integer.

Examples

```
X := Floor(-10.4);  // X is -11
X := Floor(15.98);  // X is 15
```

9.31 HideProgress

Unit: WebForms

```
procedure HideProgress
```

The **HideProgress** procedure decrements the global progress reference count, and if the reference count is 0, hides the active progress dialog. The ShowProgress procedure shows a progress dialog and increments the progress reference count.

Examples

```
HideProgress;
```

9.32 HourOf

Unit: Internal

```
function HourOf(Value: DateTime; UTC: Boolean=False): Integer
```

The **HourOf** function returns the hour number of the input parameter in local or UTC time. The return value is an Integer value between 0 (midnight) and 23 (11:00 PM).

Examples

```
X := HourOf(Time);  // X is 12 (assuming a time of 12:10 PM)
```

9.33 Inc

Unit: Internal

```
procedure Inc(var Value: Integer)

procedure Inc(var Value: Integer; By: Integer)
```

The **Inc** procedure increments the Integer input parameter by 1, or by the By input parameter, if specified.

Examples

```
X := 1;
Inc(X);
Y := IntToStr(X)  // Y is '2'
```

9.34 Insert

Unit: Internal

```
procedure Insert(Value: <Type>; const Array: array of <Type>;
                 Index: Integer)

procedure Insert(const Value: array of <Type>; const Array: array of <Type>;
                 Index: Integer)
```

The **Insert** procedure inserts a new value (or array of values) into the Array input parameter. The Index input parameter specifies where the insertion will take place. The Value input parameter must be type-compatible with the Array input parameter that it is being inserted into.

Note
This procedure cannot be used with strings in Elevate Web Builder. Strings are immutable in JavaScript, and therefore cannot be modified in-place using procedures such as this. For more information on these types, please see the Types topic.

Examples

```
X := [10,20,30,40];
Insert(35, X, 3); // X is [10,20,30,35,40] after the Insert call

X := [10,20,30,40];
Y := [21,22,23,24,25];
Insert(Y, X, 2); // X is [10,20,21,22,23,24,25,30,40] after the Insert call
```

9.35 IntToHex

Unit: Internal

```
function IntToHex(Value: Integer; Digits: Integer): String
```

The **IntToHex** function returns a formatted hexadecimal string for the Integer input parameter. The Digits input parameter indicates the minimum length of the String return value.

Examples

```
X := IntToHex(1052);  // X is '041C'
```

9.36 IntToStr

Unit: Internal

```
function IntToStr(Value: Integer): String
```

The **IntegerToStr** function returns a formatted string for the Integer input parameter. The return value is a String value.

Examples

```
X := IntToStr(-102);  // X is '-102'
```

9.37 ISOStrToDateTime

Unit: Internal

```
function ISOStrToDateTime(const Value: String): DateTime
```

The **ISOStrToDateTime** function returns a date-time value for the ISO-8601-formatted date-time string input parameter. The return value is a DateTime value.

Examples

```
X := DateTimeToStr(ISOStrToDateTime('2015-08-15T16:40:31.601Z'));  // X is
      '8/15/2015 12:40 PM'
```

9.38 Join

Unit: Internal

```
function Join(const Array: array of String; const Separator: String): String

function Join(const Array: array of String): String
```

The **Join** function builds a new string from the elements in the Array string array input parameter. The Separator input parameter is optional. If the Separator input parameter is specified, then the return value is a String value that contains all string elements from the array separated by the Separator input parameter. If the Separator input parameter is not specified, then the return value is a String value that contains all string elements from the array.

Examples

```
X := Join(['Hello,','my','name','is','Jim'], ' ');  // X is 'Hello, my name
      is Jim'
```

9.39 Length

Unit: Internal

```
function Length(const Value: String): Integer

function Length(const Value: array of <Type>): Integer
```

The **Length** function returns the length of the String or array input parameter. The return value is an Integer value.

Examples

```
X := Length('How long is this string');  // X is 23
```

9.40 Ln

Unit: Internal

```
function Ln(Value: Double): Double

function Ln(Value: Integer): Double
```

The **Ln** function returns the natural logarithm of the input parameter, which must be greater than 0. The return value is a Double value.

Examples

```
X := Ln(0.523);  // X is -0.6481738149172141
```

9.41 LocaleCompareStr

Unit: Internal

```
function LocaleCompareStr(const A, B: String): Integer
```

The **LocaleCompareStr** function compares the A input parameter string with the B input parameter string with case-sensitivity. The comparison uses the browser's current locale setting to compare the two strings. The return value is an Integer value of -1 if A is less than B, 0 if A is equal to B, and 1 if A is greater than B.

Examples

```
X := LocaleCompareStr('Absolute', 'Baseball');  // X is -1
```

9.42 LocaleCompareText

Unit: Internal

```
function LocaleCompareText(const A, B: String): Integer
```

The **LocaleCompareText** function compares the A input parameter string with the B input parameter string without case-sensitivity. The comparison uses the browser's current locale setting to compare the two strings. The return value is an Integer value of -1 if A is less than B, 0 if A is equal to B, and 1 if A is greater than B.

Examples

```
X := LocaleCompareText('Absolute', 'ABSOLUTE');  // X is 0
```

9.43 LocaleLowerCase

Unit: Internal

```
function LocaleLowerCase(const Value: String): String
```

The **LocaleLowerCase** function returns the Value input parameter with all characters converted to their lower-case representation. The browser's current locale setting is used to perform this conversion. The return value is a String value.

Examples

```
X := LocaleLowerCase('Hello World');  // X is 'hello world'
```

9.44 LocaleSameStr

Unit: Internal

```
function LocaleSameStr(const A, B: String): Boolean
```

The **LocaleSameStr** function compares the A input parameter string with the B input parameter string with case-sensitivity. The comparison uses the browser's current locale setting to compare the two strings. The return value is a Boolean value of True if A is equal to B, and False if A is not equal to B.

Examples

```
X := LocaleSameStr('Absolute', 'Baseball');  // X is False
```

9.45 LocaleSameText

Unit: Internal

```
function LocaleSameText(const A, B: String): Boolean
```

The **LocaleSameText** function compares the A input parameter string with the B input parameter string without case-sensitivity. The comparison uses the browser's current locale setting to compare the two strings. The return value is a Boolean value of True if A is equal to B, and False if A is not equal to B.

Examples

```
X := LocaleSameStr('Absolute', 'ABSOLUTE');  // X is True
```

9.46 LocaleUpperCase

Unit: Internal

```
function LocaleUpperCase(const Value: String): String
```

The **LocaleUpperCase** function returns the Value input parameter with all characters converted to their upper-case representation. The browser's current locale setting is used to perform this conversion. The return value is a String value.

Examples

```
X := LocaleUpperCase('Hello World');  // X is 'HELLO WORLD'
```

9.47 LowerCase

Unit: Internal

```
function LowerCase(const Value: String): String
```

The **LowerCase** function returns the Value input parameter with all characters converted to their lower-case representation. The browser's current locale setting is not used to perform this conversion. The return value is a String value.

Examples

```
X := LowerCase('Hello World');  // X is 'hello world'
```

9.48 Max

Unit: Internal

```
function Max(A,B: Integer): Integer

function Max(A,B: Double): Double
```

The **Max** function returns the greater of the two input parameters. If A is greater than B, then A is returned. If B is greater than A, then B is returned. The return value is the same type as the input parameters.

Examples

```
X := Max(100,2);  // X is 100
```

9.49 MessageDlg

Unit: WebForms

```
procedure MessageDlg(const Msg: String;
                     const DlgCaption: String;
                     DlgType: TMsgDlgType;
                     const Buttons: TMsgDlgBtns;
                     MsgDlgResult: TMsgDlgResultEvent=nil;
                     CloseButton: Boolean=False;
                     OverlayColor: TColor=clBlack;
                     OverlayOpacity: Double=20;
                     AnimationStyle: TAnimationStyle=asNone;
                     AnimationDuration: Integer=0)

procedure MessageDlg(const Msg: String;
                     const DlgCaption: String;
                     DlgType: TMsgDlgType;
                     const Buttons: TMsgDlgBtns;
                     DefaultButton: TMsgDlgBtn;
                     MsgDlgResult: TMsgDlgResultEvent=nil;
                     CloseButton: Boolean=False;
                     OverlayColor: TColor=clBlack;
                     OverlayOpacity: Double=20;
                     AnimationStyle: TAnimationStyle=asNone;
                     AnimationDuration: Integer=0)
```

The **MessageDlg** procedure shows a modal message dialog.

The Msg parameter indicates the message to show.

The DlgCaption parameter indicates the caption of the dialog.

The DlgType parameter indicates the type of dialog to show.

The Buttons parameter indicates the array of button types to use on the dialog.

The DefaultButton parameter indicates which button should have focus when the dialog is first shown.

The MsgDlgResult parameter is a method reference that represents the event handler that will be called when the message dialog is closed.

The CloseButton parameter indicates whether the message dialog should contain a close button.

The OverlayColor parameter indicates the color to use for the shadow overlay effect that is used over the application desktop to indicate that a modal form is in effect.

The OverlayOpacity parameter indicates the percentage of opacity to use for the shadow overlay effect that is used over the application desktop to indicate that a modal form is in effect.

The AnimationStyle and AnimationDuration parameters indicate the type/duration of animation to use when showing the message dialog.

Examples

```
MessageDlg('Are you sure that you want to delete this record?',
           'Please Confirm',mtConfirmation,[mbYes,mbNo],mbNo,CheckDelete,
      True);
```

9.50 Min

Unit: Internal

```
function Min(A,B: Integer): Integer

function Min(A,B: Double): Double
```

The **Min** function returns the lesser of the two input parameters. If A is less than B, then A is returned. If B is less than A, then B is returned. The return value is the same type as the input parameters.

Examples

```
X := Min(100,2);  // X is 2
```

9.51 MinuteOf

Unit: Internal

```
function MinuteOf(Value: DateTime; UTC: Boolean=False): Integer
```

The **MinuteOf** function returns the minute number of the input parameter in local or UTC time. The return value is an Integer value between 0 and 59.

Examples

```
X := MinuteOf(Time);  // X is 10 (assuming a time of 12:10 PM)
```

9.52 MonthOf

Unit: Internal

```
function MonthOf(Value: DateTime; UTC: Boolean=False): Integer
```

The **MonthOf** function returns the month number of the input parameter in local or UTC time. The return value is an Integer value.

Examples

```
X := MonthOf(Date);  // X is 2 (assuming a date of 02/13/2012)
```

9.53 MSecondOf

Unit: Internal

```
function MSecondOf(Value: DateTime; UTC: Boolean=False): Integer
```

The **MSecondOf** function returns the millisecond number of the input parameter in local or UTC time. The return value is an Integer value between 0 and 59.

Examples

```
X := MSecondOf(Time);  // X is 247 (assuming a time of 12:10:20.247 PM)
```

9.54 Now

Unit: Internal

```
function Now: DateTime
```

The **Now** function returns the current date and time. The return value is a DateTime value.

Examples

```
X := DateTimeToStr(Now);  // X is '2/13/2012 12:10 PM'
```

9.55 Ord

Unit: Internal

```
function Ord(Value: Char): Integer

function Ord(Value: Boolean): Integer
```

The **Ord** function returns the Unicode code point for a Char input parameter, 0 for a False Boolean input parameter, and 1 for a True Boolean input parameter. The return value is an Integer.

Examples

```
X := Ord('Ü');  // X is 220

X := Ord(True);  // X is 1
```

9.56 Pad

Unit: WebCore

```
function Pad(const Value: String; PadLen: Integer;
             PadChar: Char=' '): String
```

The **Pad** function returns the Value input parameter padded to the length specified by the PadLen input parameter. The optional PadChar input parameter specifies the character to be used for padding the Value input parameter, and defaults to a space (' ') character. The return value is a String value.

> **Note**
> The padding is inserted on the left side of the string.

Examples

```
X := Pad('100', 10);  // X is '       100'
```

9.57 Pi

Unit: Internal

```
function Pi: Double
```

The **Pi** function returns the mathematical constant pi, or the ratio of the circumference of a circle to its diameter. The return value is a Double value that is approximately 3.141592653589793.

Examples

```
X := Pi;  // X is 3.141592653589793
```

9.58 ParseXML

Unit: WebCore

```
function ParseXML(const Value: String): TDocument
```

The **ParseXML** function parses the XML string input parameter and returns a TDocument class instance.

> **Note**
> Please refer to the WebDOM unit source code for the declaration of the TNode, TDocument, and TNodeList classes and their various properties.

Examples

```
var
   TempXML: String;
   TempDocument: TDocument;
   TempNodes: TNodeList;
begin
   TempXML := '<a><b><c><username>testuser</username></c></b></a>';
   TempDocument := ParseXML(TempXML);
   TempNodes:=TempDocument.getElementsByTagName('username');
   ShowMessage(IntToStr(TempNodes.length)); // Number of nodes
   ShowMessage(TempNodes[0].firstChild.nodeValue); // Get text node
   TempXML := SerializeXML(TempDocument);
   ShowMessage(TempXML);
end;
```

9.59 Pos

Unit: Internal

```
function Pos(const SearchValue: String; const Value: String;
             Index: Integer=1): Integer
```

The **Pos** function returns the position of the SearchValue input parameter in the Value input parameter. The optional Index parameter specifies the starting index of the search and, if not specified, the search will start at the first character of the Value input parameter. The return value is an Integer value of 0 if the SearchValue input parameter was not found, or the index of the SearchValue if it was found.

Examples

```
X := Pos(' ', 'Whereisthe spaceinthisstring');  // X is 11
```

9.60 Power

Unit: Internal

```
function Power(X, Y: Double): Double

function Power(X, Y: Integer): Double
```

The **Power** function returns the X input parameter raised to the power specified by the Y input parameter. The return value is a Double value.

Examples

```
X := Power(0.523, 4);  // X is 0.07481811384100001
```

9.61 Radians

Unit: Internal

```
function Radians(Value: Double): Double

function Radians(Value: Integer): Double
```

The **Radians** function converts the input parameter, which is an angle specified in degrees, to radians. The return value is a Double value.

Examples

```
X := Radians(5294.01543544978);  // X is 92.398
```

9.62 Random

Unit: Internal

```
function Random(AFrom: Integer=0; ATo=<MaxInt>): Integer
```

The **Random** function returns a pseudorandom number greater than or equal to the AFrom parameter, if provided, and less than or equal to the ATo parameter, if provided. The default AFrom paramter value is 0, and the default ATo parameter value is the maximum integer value. At runtime, the maximum integer value is 9007199254740991, and at design-time the maximum integer value is 9223372036854775807. The return value is an Integer value.

Examples

```
X := Random;  // X is 7534176611 (pseudorandom value)
X := Random(0,1000);  // X is 269 (pseudorandom value)
```

9.63 Round

Unit: Internal

```
function Round(Value: Double): Integer

function Round(Value: Integer): Integer
```

The **Round** function returns the closest integer to the value of the input parameter using the "round half up" method. The return value is an Integer.

Examples

```
X := Round(-10.4);  // X is -10
X := Round(15.5);  // X is 16
```

9.64 QuotedStr

Unit: WebCore

```
function QuotedStr(const Value: String;
                   QuoteChar: Char=SINGLE_QUOTE): String
```

The **QuotedStr** function adds the specified quote character to the start and end of the input parameter. In addition, any characters in the input parameter that match the specified quote character are escaped so that they are properly interpreted as embedded quote characters. The return value is the transformed input parameter.

Examples

```
Y := 'Absolute';
X := QuotedStr(Y);  // X is 'Absolute'

Y := 'It''s';
X := QuotedStr(Y);  // X is 'Its''s'
```

9.65 SameStr

Unit: Internal

```
function SameStr(const A, B: String): Boolean
```

The **SameStr** function compares the A input parameter string with the B input parameter string with case-sensitivity. The comparison is locale-insensitive. The return value is a Boolean value of True if A is equal to B, and False if A is not equal to B.

Examples

```
X := SameStr('Absolute', 'Baseball');  // X is False
```

9.66 SameText

Unit: Internal

```
function SameText(const A, B: String): Boolean
```

The **SameText** function compares the A input parameter string with the B input parameter string without case-sensitivity. The comparison is locale-insensitive. The return value is a Boolean value of True if A is equal to B, and False if A is not equal to B.

Examples

```
X := SameStr('Absolute', 'ABSOLUTE');  // X is True
```

9.67 SecondOf

Unit: Internal

```
function SecondOf(Value: DateTime; UTC: Boolean=False): Integer
```

The **SecondOf** function returns the second number of the input parameter in local or UTC time. The return value is an Integer value between 0 and 59.

Examples

```
X := SecondOf(Time);  // X is 20 (assuming a time of 12:10:20 PM)
```

9.68 SerializeXML

Unit: WebCore

```
function SerializeXML(Document: TDocument): String
```

The **SerializeXML** function converts the XML nodes present in the TDocument instance parameter into a string. The return value is a String value.

Note
 Please refer to the WebDOM unit source code for the declaration of the TNode, TDocument, and TNodeList classes and their various properties.

Examples

```
var
   TempXML: String;
   TempDocument: TDocument;
   TempNodes: TNodeList;
begin
   TempXML := '<a><b><c><username>testuser</username></c></b></a>';
   TempDocument := ParseXML(TempXML);
   TempNodes:=TempDocument.getElementsByTagName('username');
   ShowMessage(IntToStr(TempNodes.length)); // Number of nodes
   ShowMessage(TempNodes[0].firstChild.nodeValue); // Get text node
   TempXML := SerializeXML(TempDocument);
   ShowMessage(TempXML);
end;
```

9.69 SetLength

Unit: Internal

```
procedure SetLength(var Value: array of <Type>)
```

The **SetLength** procedure sets the length of the Value array input parameter. If extending the length of an array, all new elements are automatically set to nil for string, object, or method arrays, NaN (not a number) for numeric (Integer or Double) arrays, and False for boolean arrays.

> **Warning**
> Arrays that are declared but not assigned a value are nil (Assigned function returns False) and not initialized. The SetLength procedure is one way of initializing them, even if they are initialized to a length of 0.

Examples

```
var
   X: array of String;
   I: Integer;
begin
   SetLength(X, 10);  // X now has a length of 10, but each element is still
      nil
   for I := 0 to Length(X)-1 do
      X[I] := '';  //  Initialize each array element with an empty string
end;
```

9.70 ShowMessage

Unit: WebForms

```
procedure ShowMessage(const Msg: String;
                      const DlgCaption: String='';
                      AnimationStyle: TAnimationStyle=asNone;
                      AnimationDuration: Integer=0)
```

The **ShowMessage** procedure shows a simple modal message dialog. The Msg parameter indicates the message to show. The DlgCaption parameter is optional and indicates the caption of the dialog. The AnimationStyle and AnimationDuration parameters are optional, and indicate the type/duration of animation to use when showing the message dialog.

Examples

```
ShowMessage('An error has occurred !','Error');

ShowMessage('Hello world !','Hi !',asQuadEaseOut,500);
```

9.71 ShowProgress

Unit: WebForms

```
procedure ShowProgress(const Msg: String;
                       AnimationStyle: TAnimationStyle=asNone;
                       AnimationDuration: Integer=0)
```

The **ShowProgress** procedure shows a modal progress dialog and increments the global progress reference count. The HideProgress procedure decrements the reference count and hides any active progress dialog. The AnimationStyle and AnimationDuration parameters are optional, and indicate the type/duration of animation to use when showing the progress dialog.

Examples

```
ShowProgress('Loading customers...');
```

9.72 Sin

Unit: Internal

```
function Sin(Value: Double): Double

function Sin(Value: Integer): Double
```

The **Sin** function returns the sine of the input parameter, which is an angle specified in radians. To convert an angle from degrees to radians, use the Radians function. The return value is a Double value between -1 and 1.

Examples

```
X := Sin(0.23290);  // X is 0.2308001934780994
```

9.73 Split

Unit: Internal

```
function Split(const Value: String; const Separator: String): array of String

function Split(const Value: String; const Separator: String;
               MaxLength: Integer): array of String
```

The **Split** function builds a new string array by splitting the Value input parameter using the Separator input parameter. The optional MaxLength input parameter specifies the maximum length of the string array. The return value is an array of String values that **does not** include the specified Separator string.

Examples

```
X := Split('Hello, my name is Jim', ' ');  // X is ['Hello,','my','name','is',
      'Jim']
```

9.74 Sqrt

Unit: Internal

```
function Sqrt(Value: Double): Double

function Sqrt(Value: Integer): Double
```

The **Sqrt** function returns the square root of the input parameter. The return value is a Double value.

Examples

```
X := Sqrt(154);  // X is 12.409673645990857
```

9.75 StrReplace

Unit: WebCore

```
function StrReplace(const Value: String; const SearchValue: String;
                    const ReplaceValue: String;
                    ReplaceAll: Boolean=False;
                    CaseInsensitive: Boolean=False): String
```

The **StrReplace** function searches for the SearchValue input parameter in the Value input parameter and replaces it with the ReplaceValue input parameter. If the optional ReplaceAll input parameter is True, then all occurrences of the SearchValue input parameter are replaced with the ReplaceValue input parameter. If the optional CaseInsensitive input parameter is True, then the search for the SearchValue input parameter will be case-insensitive. The return value is the modified String value.

Examples

```
X := StrReplace('abcdefghijk', 'd', ' ', True, True);  // X is 'abc efghijk'
```

9.76 StrToBool

Unit: WebCore

```
function StrToBool(const Value: String): Boolean
```

The **StrToBool** function returns True if the input parameter is 'True' (case-insensitive), and False if the input parameter is 'False' (case-insensitive also). The return value is a Boolean value.

Examples

```
X := StrToBool('True');  // X is True
```

9.77 StrToDate

Unit: WebCore

```
function StrToDate(const Value: String; UTC: Boolean=False): DateTime
```

The **StrToDate** function converts the formatted local or UTC date string input parameter into its native value. The required format of the string is determined by the TFormatSettings ShortDateFormat property. The return value is a DateTime value.

Examples

```
A := StrToDate('2/13/2012');
X := DateToStr(A);  // X is '2/13/2012'
```

9.78 StrToDateTime

Unit: WebCore

```
function StrToDateTime(const Value: String; UTC: Boolean=False): DateTime
```

The **StrToDateTime** function converts the formatted local or UTC date and time string input parameter into its native value. The required format of the string is determined by the TFormatSettings ShortDateFormat and ShortTimeFormat properties. The return value is a DateTime value.

Examples

```
A := StrToDateTime('2/13/2012 12:10 PM');
X := DateTimeToStr(A);  // X is '2/13/2012 12:10 PM'
```

9.79 StrToDouble

Unit: Internal

```
function StrToDouble(const Value: String): Double
```

The **StrToDouble** function converts the formatted string input parameter into its native value. The decimal separator used in the formatted string is **always** required to be a period (.). The return value is a Double value.

Examples

```
A := StrToDouble('1200.548');  // X is 1200.548
```

9.80 StrToFloat

Unit: Internal

```
function StrToFloat(const Value: String): Double
```

The **StrToFloat** function converts the formatted string input parameter into its native value. The decimal separator used in the formatted string is **always** a period (.). The return value is a Double value.

Examples

```
A := StrToFloat('1200.548');
X := FloatToStr(A);  // X is '1200.548'
```

9.81 StrToInt

Unit: Internal

```
function StrToInt(const Value: String): Int
```

The **StrToInt** function converts the formatted string input parameter into its native value. The return value is an Integer value.

Examples

```
X := StrToInt('-102');  // X is -102
```

9.82 StrToTime

Unit: WebCore

```
function StrToTime(const Value: String; UTC: Boolean=False): DateTime
```

The **StrToTime** function converts the formatted local or UTC time string input parameter into its native value. The required format of the string is determined by the TFormatSettings ShortTimeFormat properties. The return value is a DateTime value.

Examples

```
A := StrToTime('12:10 PM');
X := TimeToStr(A);  // X is '12:10 PM'
```

9.83 Tan

Unit: Internal

```
function Tan(Value: Double): Double

function Tan(Value: Integer): Double
```

The **Tan** function returns the tangent of the input parameter, which is an angle specified in radians. To convert an angle from degrees to radians, use the Radians function. The return value is a Double value.

Examples

```
X := Tan(0.23290);  // X is 0.23720443648121617
```

9.84 Time

Unit: Internal

```
function Time: DateTime
```

The **Time** function returns the current time. The return value is a DateTime value.

Examples

```
X := TimeToStr(Time);  // X is '12:10 PM'
```

9.85 TimeToStr

Unit: WebCore

```
function TimeToStr(Value: DateTime; UTC: Boolean=False): String
```

The **TimeToStr** function returns a formatted local or UTC time string for the DateTime input parameter. The format of the string is determined by the TFormatSettings ShortTimeFormat properties. The return value is a String value.

Examples

```
A := StrToTime('12:10 PM');
X := TimeToStr(A);  // X is '12:10 PM'
```

9.86 TimeZoneOffset

Unit: Internal

```
function TimeZoneOffset(Value: DateTime): Integer
```

The **TimeZoneOffset** function returns the time zone offset for the input parameter. The return value is an Integer value that represents the time zone offset expressed in minutes.

Examples

```
X := TimeZoneOffset(Now);  // X is 240 (4 hours) for US EST during
                                   // daylight savings time (summer)
```

9.87 Trim

Unit: WebCore

```
function Trim(const Value: String): String

function Trim(const Value: String; TrimChar: Char): String
```

The **Trim** function returns the Value input parameter with both leading and trailing "space" characters removed. The first version of this function trims all leading and trailing characters that are less than or equal to the space (#32) character from the string. The second version of this function allows the developer to specify the character that should be trimmed from the Value parameter. The return value is a String value.

Examples

```
X := Trim('  Hello World  ');  // X is 'Hello World'
```

9.88 Trunc

Unit: WebCore

```
function Trunc(Value: Double): Integer

function Trunc(Value: Integer): Integer
```

The **Trunc** function returns the closest (towards 0) integer from the value of the input parameter. The return value is an Integer.

Examples

```
X := Trunc(-10.4);  // X is -10
X := Trunc(15.98);  // X is 15
```

9.89 UpperCase

Unit: Internal

```
function UpperCase(const Value: String): String
```

The **UpperCase** function returns the Value input parameter with all characters converted to their uppercase representation. The browser's current locale setting is not used to perform this conversion. The return value is a String value.

Examples

```
X := UpperCase('Hello World');  // X is 'HELLO WORLD'
```

9.90 WeekDayOf

Unit: Internal

```
function WeekDayOf(Value: DateTime; UTC: Boolean=False): Integer
```

The **WeekDayOf** function returns the week day number of the input parameter in local or UTC time. This function is ISO 8601-compliant, meaning that the week days of Monday through Sunday are represented by the values 1 through 7, respectively. The return value is an Integer value.

Examples

```
X := WeekDayOf(Date);  // X is 1 (Monday, assuming a date of 02/13/2012)
```

9.91 YearOf

Unit: Internal

```
function YearOf(Value: DateTime; UTC: Boolean=False): Integer
```

The **YearOf** function returns the year number of the input parameter in local or UTC time. The return value is an Integer value.

Examples

```
X := YearOf(Date);  // X is 2012 (assuming a date of 02/13/2012)
```

Chapter 10
Component Reference

10.1 TAbstractList Component

Unit: WebCore

Inherits From TPersistent

This class represents an abstract list and is used as the ancestor class for the TObjectList and TStrings classes. It provides the functionality for tracking changes to the list as well as dealing with batch updates to the list.

Properties	Methods	Events
	BeginUpdate	OnChanged
	EndUpdate	

TAbstractList.BeginUpdate Method

```
procedure BeginUpdate
```

Use this method to begin a batch update to the list. Batch updates are useful in situations where many changes need to be made to the list, and triggering the OnChanged event on every change would result in performance issues. This method is reference-counted and every time it is called, an internal counter is incremented. Every time the EndUpdate method is called, the counter is decremented. Once the counter reaches zero, the OnChanged event will be triggered.

TAbstractList.EndUpdate Method

```
procedure EndUpdate
```

Use this method to end a batch update to the list. Batch updates are useful in situations where many changes need to be made to the list, and triggering the OnChanged event on every change would result in performance issues. This method is reference-counted and every time it is called, an internal counter is decremented. Every time the BeginUpdate method is called, the counter is incremented. Once the counter reaches zero, the OnChanged event will be triggered.

TAbstractList.OnChanged Event

```
property OnChanged: TNotifyEvent
```

This event is triggered whenever the list is modified in any way. If a batch update is in effect via the BeginUpdate and EndUpdate methods, then this event is only triggered once the EndUpdate call is made. If multiple calls to BeginUpdate and EndUpdate are nested, then this event is only triggered once the last matching EndUpdate call is made.

10.2 TAddress Component

Unit: WebComps

Inherits From TObject

The TAddress class encapsulates the address bar (location) functionality in the web browser.

Note
The component library includes a global instance variable of this class called Address in the WebComps unit that should be used instead of creating new instances of the class.

Warning
The methods of this class, as well as assignments to the various properties, can cause the browser to navigate to a new resource location, and this will cause the current application to be unloaded. The sole exception are any assignments to the Anchor property. Changes to the Anchor property will not cause the current application to be unloaded.

Properties	Methods	Events
Anchor	Assign	OnAnchorChange
Host	Create	
HostName	Reload	
Params	Replace	
Path		
Port		
Protocol		
URL		

TAddress.Anchor Property

```
property Anchor: String
```

Specifies the anchor (#) portion for the address. If the specified anchor is different than the anchor in the current address, then the browser will trigger the event handler assigned to the OnAnchorChange event property of the global Address instance. The assigned event handler can then take specific action based upon the change.

TAddress.Host Property

```
property Host: String
```

Specifies the host (host name and port) portion for the address. If the specified host is different than the host in the current address, then the browser will unload the current application and load the resource specified by the new composed address.

TAddress.HostName Property

```
property HostName: String
```

Specifies the host name (www.mysite.com) portion for the address. If the specified host name is different than the host name in the current address, then the browser will unload the current application and load the resource specified by the new composed address.

TAddress.Params Property

```
property Params: String
```

Specifies the query parameters (?param1=100&param2=200) portion for the address. If the specified query parameters are different than the query parameters in the current address, then the browser will unload the current application and load the resource specified by the new composed address.

TAddress.Path Property

```
property Path: String
```

Specifies the path portion for the address. If the specified path is different than the path in the current address, then the browser will unload the current application and load the resource specified by the new composed address.

TAddress.Port Property

```
property Port: String
```

Specifies the port (:80, :8080, etc.) portion for the address. If the specified port is different than the port in the current address, then the browser will unload the current application and load the resource specified by the new composed address.

TAddress.Protocol Property

```
property Protocol: String
```

Specifies the protocol (normally http: or https:) portion for the address. If the specified protocol is different than the protocol in the current address, then the browser will unload the current application and load the resource specified by the new composed address.

TAddress.URL Property

```
property URL: String
```

Specifies the complete address. If the specified address is different than the current address, then the browser will unload the current application and load the resource specified by the new address.

TAddress.Assign Method

```
procedure Assign(const AURL: String)
```

Use this method to load the resource specified by the AURL parameter. Using this method will result in a new history entry for the resource in the browser's navigation history.

TAddress.Create Method

```
constructor Create
```

Use this method to create a new instance of the TAddress class.

TAddress.Reload Method

```
procedure Reload(Force: Boolean=False)
```

Use this method to reload the resource for the current address. By default, the browser will attempt to reload the resource from its cache. Setting the Force parameter to True will cause the browser to ignore its cached and reload the resource from its source (web server, file system, etc.).

TAddress.Replace Method

```
procedure Replace(const AURL: String)
```

Use this method to load the resource specified by the AURL parameter. Using this method will result in the history entry for the current address being replaced in the browser's navigation history with the address of the resource.

TAddress.OnAnchorChange Event

```
property OnAnchorChange: TNotifyEvent
```

This event is triggered whenever the anchor (#) portion of the current address changes. This is useful for navigating in your application using the browser's address bar and forward/back buttons.

10.3 TAlertLabel Component

Unit: WebLabels

Inherits From TAlertLabelControl

The TAlertLabel component represents a label control that can be used to display alerts that, by default, appear at the top of the client area of their container, with customizations such as the orientation of the caption and whether or not to show a close button.

Properties	Methods	Events
AllowClose		OnAnimationComplete
AutoHeight		OnAnimationsComplete
Background		OnClick
Border		OnClose
Caption		OnCloseQuery
Corners		OnDblClick
Cursor		OnHide
DataColumn		OnMouseDown
DataSet		OnMouseEnter
Font		OnMouseLeave
Format		OnMouseMove
Hint		OnMouseUp
Opacity		OnMove
Orientation		OnShow
Padding		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TAlertLabel.AllowClose Property

```
property AllowClose: Boolean
```

Specifies whether the close button should be shown.

TAlertLabel.AutoHeight Property

```
property AutoHeight: Boolean
```

Specifies whether the height of the alert label should be automatically set based upon the Caption, Font, and Format properties.

TAlertLabel.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TAlertLabel.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TAlertLabel.Caption Property

```
property Caption: TCaption
```

Specifies the text to be shown in the alert label control. The text can contain line feeds. The default value is ''.

TAlertLabel.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TAlertLabel.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TAlertLabel.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TAlertLabel.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TAlertLabel.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TAlertLabel.Format Property

```
property Format: TFormat
```

Specifies the content formatting to use for the control's Caption.

TAlertLabel.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TAlertLabel.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TAlertLabel.Orientation Property

```
property Orientation: TAlertOrientation
```

Specifies the orientation of the alert label caption.

TAlertLabel.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

TAlertLabel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TAlertLabel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TAlertLabel.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TAlertLabel.OnClose Event

```
property OnClose: TNotifyEvent
```

This event is triggered when the label is closed by the user via the close button, or when the Close method is called.

TAlertLabel.OnCloseQuery Event

```
property OnCloseQuery: TCloseQueryEvent
```

This event is triggered when the label is closed by the user via the close button, or when the Close method is called. Return True to allow the close to continue, or False to prevent the label from closing.

TAlertLabel.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TAlertLabel.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TAlertLabel.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TAlertLabel.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TAlertLabel.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TAlertLabel.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TAlertLabel.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TAlertLabel.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TAlertLabel.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TAlertLabel.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TAlertLabel.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TAlertLabel.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TAlertLabel.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TAlertLabel.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.4 TAlertLabelControl Component

Unit: WebLabels

Inherits From TBindableColumnControl

The TAlertLabelControl control is the base class for alert label controls, and contains all of the label functionality in the form of public methods and protected properties/events that descendant classes can use to create customized alert label controls.

Properties	Methods	Events
	Close	

TAlertLabelControl.Close Method

```
procedure Close
```

Use this method to close the label. When this method is called, the OnCloseQuery event is triggered, followed by the OnClose event. If the OnCloseQuery event handler returns True, then the label will be hidden before the OnClose event is triggered.

10.5 TAnimatedIcon Component

Unit: WebIcons

Inherits From TIconControl

The TAnimatedIcon component represents an animated icon control. An animated icon control displays a special type of icon, referenced by the Icon property, that contains a series of animation frames as a single background image. These animation frames can be oriented horizontally or vertically, and the Direction property allows you to specify the direction.

Properties	Methods	Events
Cursor	StartAnimating	OnAnimationComplete
Direction	StopAnimating	OnAnimationsComplete
Hint		OnClick
Icon		OnDblClick
Opacity		OnHide
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TAnimatedIcon.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TAnimatedIcon.Direction Property

```
property Direction: TAnimatedIconDirection
```

Specifies the direction in which the animation frames in the icon are oriented.

TAnimatedIcon.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TAnimatedIcon.Icon Property

```
property Icon: TIconProperties
```

Specifies the properties of the icon used with the control.

TAnimatedIcon.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TAnimatedIcon.StartAnimating Method

```
procedure StartAnimating
```

Use this method to begin animating the icon specified in the Icon property.

TAnimatedIcon.StopAnimating Method

```
procedure StopAnimating
```

Use this method to stop animating the icon specified in the Icon property.

TAnimatedIcon.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TAnimatedIcon.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TAnimatedIcon.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TAnimatedIcon.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TAnimatedIcon.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TAnimatedIcon.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TAnimatedIcon.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TAnimatedIcon.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TAnimatedIcon.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TAnimatedIcon.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TAnimatedIcon.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TAnimatedIcon.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TAnimatedIcon.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TAnimatedIcon.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TAnimatedIcon.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TAnimatedIcon.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TAnimatedIcon.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.6 TAnimation Component

Unit: WebUI

Inherits From TElementAttribute

The TAnimation class represents the properties of an animation. The animation properties include the style of the animation and the duration, in milliseconds, of the animation.

Properties	Methods	Events
Duration	Cancel	
Running	SetToDefault	
Style	Start	

TAnimation.Duration Property

```
property Duration: Integer
```

Specifies how long, in milliseconds, the animation should take to execute.

TAnimation.Running Property

```
property Running: Boolean
```

Specifies if the animation is currently running.

TAnimation.Style Property

```
property Style: TAnimationStyle
```

Specifies the style of the animation, which controls how the animation transforms a given UI element/control property. Currently, the supported styles include all of the standard easing transformations (including linear).

TAnimation.Cancel Method

```
procedure Cancel
```

Use this method to cancel an animation.

Warning
Do not directly call this method. It is used internally by the interface manager.

TAnimation.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the animation's properties to their default values.

TAnimation.Start Method

```
procedure Start(EndValue: Integer)
```

Use this method to start an animation.

Warning
Do not directly call this method. It is used internally by the interface manager.

10.7 TAnimations Component

Unit: WebUI

Inherits From TElementAttribute

The TAnimations class represents the properties that can be animationed for a UI element or control. These properties currently include the Left, Top, Width, Height, Opacity, and Visible properties.

> **Note**
> When an animation is specified for a property, then that animation is applied **whenever** the property changes.

Properties	Methods	Events
Height	SetToDefault	
Left		
Opacity		
Top		
Visible		
Width		

TAnimations.Height Property

```
property Height: TAnimation
```

Specifies the animation properties for the Height property.

TAnimations.Left Property

```
property Left: TAnimation
```

Specifies the animation properties for the Left property.

TAnimations.Opacity Property

```
property Opacity: TAnimation
```

Specifies the animation properties for the Opacity property.

TAnimations.Top Property

```
property Top: TAnimation
```

Specifies the animation properties for the Top property.

TAnimations.Visible Property

```
property Visible: TAnimation
```

Specifies the animation properties for the Visible property.

TAnimations.Width Property

```
property Width: TAnimation
```

Specifies the animation properties for the Width property.

TAnimations.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset all animation properties to their default values.

10.8 TApplication Component

Unit: WebForms

Inherits From TComponent

The TApplication component represents a visual application and provides properties and methods for dealing with the application surface, forms, and global error handling. An instance of the TApplication component called **Application** is automatically created by the component library at application startup, so further instances of the TApplication component should not be created.

Properties	Methods	Events
AutoFocus	CreateDatabase	OnError
IdleTimeout	CreateForm	OnIdle
InertiaScrollDuration	Run	
InertiaScrollStyle		
InertiaScrollThreshhold		
IsAndroid		
IsIOS		
IsWindowsPhone		
LoadProgress		
MainForm		
Surface		
Title		
TouchScrollThreshhold		
Viewport		

TApplication.AutoFocus Property

```
property AutoFocus: Boolean
```

Specifies whether an application should automatically set focus to focusable controls when showing forms, as well as restoring focus to the last-focused control when hiding forms. This property is set to True, by default, with desktop browsers, and to False, by default, with mobile browsers on Android and iOS.

TApplication.IdleTimeout Property

```
property IdleTimeout: Integer
```

Specifies the time, in seconds, that the application should wait on user input (keypresses, mouse clicks, or touches) before triggering the OnIdle event. This is useful for functionality such as making sure that any authentication information cached for the current user is discarded after a certain period of inactivity, thus forcing the user to login again when interaction with the application is resumed. The default value is 300 seconds, or 5 minutes.

Note
Mouse movement alone is not enough to reset the idle timeout. The user must specifically press a key or mouse button, or touch the surface of the screen.

TApplication.InertiaScrollDuration Property

```
property InertiaScrollDuration: Integer
```

Specifies the total amount of time, in milliseconds, that inertia scrolling will take place after the user has lifted their finger from the touch surface. The default value is 1950 milliseconds.

TApplication.InertiaScrollStyle Property

```
property InertiaScrollStyle: TAnimationStyle
```

Specifies the type of animation to use for performing inertia scrolling after the user has lifted their finger from the touch surface. The animation type determines how the scroll velocity is adjusted over the entire InertiaScrollDuration. The default value is asQuadEaseOut.

TApplication.InertiaScrollThreshhold Property

```
property InertiaScrollThreshhold: Integer
```

Specifies the finger movement velocity, in pixels per second, required on the touch surface before a touch movement will result in inertia scrolling after the user has lifted their finger from the touch surface. The default value is 10 pixels per second.

TApplication.IsAndroid Property

```
property IsAndroid: Boolean
```

Indicates whether the platform running the application is the Android platform.

TApplication.IsIOS Property

```
property IsIOS: Boolean
```

Indicates whether the platform running the application is the IOS platform.

TApplication.IsWindowsPhone Property

```
property IsWindowsPhone: Boolean
```

Indicates whether the platform running the application is the Windows Phone platform.

TApplication.LoadProgress Property

```
property LoadProgress: Boolean
```

Specifies whether load progress should be shown during the initialization and loading of an application. This is useful for applications that have many auto-create forms that may take some time to create at application startup.

TApplication.MainForm Property

```
property MainForm: TFormControl
```

Indicates the main form for the application, which is the first auto-created form for the project. This property can be modified at design-time via the **Forms** page of the Project Options dialog. This property cannot be modified at run-time.

TApplication.Surface Property

```
property Surface: TSurface
```

Contains a reference to the application's surface, which acts as the parent control to all forms and controls in a visual application.

TApplication.Title Property

```
property Title: String
```

Indicates the title of the application, which is the descriptive name that appears in the web browser for the application's tab or page. This property can be modified at design-time via the **Application** page of the Project Options dialog, and can also be modified at run-time to specify a different title.

TApplication.TouchScrollThreshhold Property

```
property TouchScrollThreshhold: Integer
```

Specifies the amount of finger movement, in pixels, required on the touch surface before a touch movement is considered a scroll movement. The default value is 4 pixels.

TApplication.Viewport Property

```
property Viewport: TViewport
```

Contains a reference to the application's viewport, which provides information about the dimensions of the browser window or container and allows the developer to specify how the applicaton surface should be scrolled within the bounds of the browser window.

TApplication.CreateDatabase Method

```
procedure CreateDatabase(ADatabaseClass: TDatabaseClass)
```

This method is called during application startup to create any databases marked for auto-creation, and should not be called manually. The list of auto-created forms and databases can be modified at design-time via the **Forms and Databases** page of the Project Options dialog.

TApplication.CreateForm Method

```
procedure CreateForm(AFormClass: TFormControlClass)
```

This method is called during application startup to create any forms marked for auto-creation, and should not be called manually. The list of auto-created forms and databases can be modified at design-time via the **Forms and Databases** page of the Project Options dialog.

TApplication.Run Method

```
procedure Run(const AMainFormName: String='')
```

This method is automatically called by the framework during application startup, and should not be called manually.

TApplication.OnError Event

```
property OnError: TErrorEvent
```

This event is triggered when an exception occurs in a visual application and is not handled by any local try..exception blocks in the code.

TApplication.OnIdle Event

```
property OnIdle: TNotifyEvent
```

This event is triggered when the application's IdleTimeout property has been exceeded.

Note
This event is only fired once per period of inactivity, and is not fired again until mouse/touch/keyboard activity or a modification to the IdleTimeout property causes it to be reset.

10.9 TAudio Component

Unit: WebMedia

Inherits From TMediaControl

The TAudio control encapsulates the HTML5 audio support available in web browsers. With the TAudio control, you can handle most aspects of audio loading and playback.

> **Note**
> This control is a visual control because it can, optionally, display a UI for controlling playback, volume, etc.

Properties	Methods	Events
AutoPlay		OnAbort
CurrentTime		OnAnimationComplete
Cursor		OnAnimationsComplete
DataColumn		OnCanPlay
DataSet		OnCanPlayThrough
DefaultPlaybackRate		OnClick
Duration		OnDblClick
Ended		OnDurationChange
Hint		OnEmptied
Loop		OnEnded
Muted		OnError
NetworkState		OnHide
Opacity		OnLoadedData
Paused		OnLoadedMetadata
PlaybackRate		OnLoadStart
Preload		OnMouseDown
ReadyState		OnMouseEnter
Seeking		OnMouseLeave
ShowControls		OnMouseMove
SourceURL		OnMouseUp
Volume		OnMove

		OnPause
		OnPlay
		OnPlaying
		OnProgress
		OnRateChange
		OnSeeked
		OnSeeking
		OnShow
		OnSize
		OnStalled
		OnSuspend
		OnTimeUpdate
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart
		OnVolumeChange
		OnWaiting

TAudio.AutoPlay Property

```
property AutoPlay: Boolean
```

Specifies that the audio should begin playing as soon as enough data has been loaded to allow playback. The default value is False.

TAudio.CurrentTime Property

```
property CurrentTime: Double
```

Indicates the current playback time, in seconds. Setting this property to a new value will cause the audio to skip to the specified time.

TAudio.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TAudio.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TAudio.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TAudio.DefaultPlaybackRate Property

```
property DefaultPlaybackRate: Double
```

Specifies the default playback rate, with 1 being normal playback, less than 1 being slower playback, and greater than 1 being faster playback. The default value is 1.

Note

The volume will normally be automatically muted when playing audio faster or slower than the normal playback rate.

TAudio.Duration Property

```
property Duration: Double
```

Indicates the length of the audio in seconds. Add an event handler for the OnDurationChange event to detect when the duration has been determined for the current audio being loaded/played. If the duration has not been determined, this property will return 0.

TAudio.Ended Property

```
property Ended: Boolean
```

Indicates that the end of the audio has been reached.

TAudio.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TAudio.Loop Property

```
property Loop: Boolean
```

Specifies that the audio playback should automatically restart at the beginning once the end has been reached. The default value is False.

TAudio.Muted Property

```
property Muted: Boolean
```

Specifies that the playback volume should be muted. The default valuse is False.

TAudio.NetworkState Property

```
property NetworkState: TMediaNetworkState
```

Indicates the network state of the audio loading/playback.

TAudio.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TAudio.Paused Property

```
property Paused: Boolean
```

Indicates that audio playback is paused, either by the user pausing the audio via the user interface when the ShowControls property is True, or by the application calling the Pause method. The default value is False.

TAudio.PlaybackRate Property

```
property PlaybackRate: Double
```

Specifies the playback rate, with 1 being normal playback, less than 1 being slower playback, and greater than 1 being faster playback. The default value is 1.

Note
The volume will normally be automatically muted when playing audio faster or slower than the normal playback rate.

TAudio.Preload Property

```
property Preload: TMediaPreload
```

Specifies how much of the current audio data should be loaded before playback begins.

TAudio.ReadyState Property

```
property ReadyState: TMediaReadyState
```

Indicates whether the audio is ready for playback, and if so, a general description of what audio data has been loaded.

TAudio.Seeking Property

```
property Seeking: Boolean
```

Indicates that audio is switching to a new playback location, either by the user changing the playback location in the audio via the user interface when the ShowControls property is True, or by the application setting the CurrentTime property.

TAudio.ShowControls Property

```
property ShowControls: Boolean
```

Specifies whether the control should show the native user interface for the audio being played.

TAudio.SourceURL Property

```
property SourceURL: String
```

Specifies the URL of the audio to be loaded into the control. Whenever this property is changed, the existing audio is cleared and the new audio will start downloading from the web server. Please review the events available for this control in order to get more information on detecting and handling the loading/playback of the audio.

TAudio.Volume Property

```
property Volume: Integer
```

Specifies the playback volume of the audio. The volume can be set between 0 and 100.

TAudio.OnAbort Event

```
property OnAbort: TNotifyEvent
```

This event is triggered whenever the media control has stopped loading data for the current media. This is normally caused by the user requesting such an action via the user interface when the ShowControls property is True.

TAudio.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TAudio.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TAudio.OnCanPlay Event

```
property OnCanPlay: TNotifyEvent
```

This event is triggered whenever the media control has loaded enough data to begin playback. However, additional data loading may be required as playback continues.

TAudio.OnCanPlayThrough Event

```
property OnCanPlayThrough: TNotifyEvent
```

This event is triggered whenever the media control has loaded enough data to begin playback and (probably) play the media until the end without needing to load any additional data.

TAudio.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TAudio.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TAudio.OnDurationChange Event

```
property OnDurationChange: TNotifyEvent
```

This event is triggered whenever the duration of the media changes, which normally occurs when loading new media into the control by modifying the SourceURL property.

TAudio.OnEmptied Event

```
property OnEmptied: TNotifyEvent
```

This event is triggered whenever an error or abort has caused the NetworkState property to revert to the mnsEmpty state.

TAudio.OnEnded Event

```
property OnEnded: TNotifyEvent
```

This event is triggered whenever playback has stopped because the end of the media has been reached.

TAudio.OnError Event

```
property OnError: TNotifyEvent
```

This event is triggered whenever an error has prevented the media from being loaded properly.

TAudio.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TAudio.OnLoadedData Event

```
property OnLoadedData: TNotifyEvent
```

This event is triggered whenever the media control has loaded enough data for the current playback location.

TAudio.OnLoadedMetadata Event

```
property OnLoadedMetadata: TNotifyEvent
```

This event is triggered whenever the media control has loaded the metadata, including the duration and dimensions, for the current media.

TAudio.OnLoadStart Event

```
property OnLoadStart: TNotifyEvent
```

This event is triggered whenever the media control starts loading data for the current media.

TAudio.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TAudio.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TAudio.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TAudio.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TAudio.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TAudio.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TAudio.OnPause Event

```
property OnPause: TNotifyEvent
```

This event is triggered whenever the media playback is paused.

TAudio.OnPlay Event

```
property OnPlay: TNotifyEvent
```

This event is triggered whenever the media playback is started/resumed.

TAudio.OnPlaying Event

```
property OnPlaying: TNotifyEvent
```

This event is triggered whenever media playback has actually started.

Note
This event is slightly different from the OnPlay event, which only indicates that the user or application **requested** playback to start/resume. This event may be triggered multiple times during playback, especially if playback needs to stop in order to allow more media data to be loaded, which can be the case with slower network connections.

TAudio.OnProgress Event

```
property OnProgress: TNotifyEvent
```

This event is triggered whenever the current media is being loaded.

Note
This event is typically fired several times per second in most web browsers, so be very careful about how time-consuming any event handlers are for this event.

TAudio.OnRateChange Event

```
property OnRateChange: TNotifyEvent
```

This event is triggered whenever the playback rate of the media control has changed for the current media. This is caused by the user requesting such an action via the user interface when the ShowControls property is True, or when the application modifies the PlaybackRate property.

TAudio.OnSeeked Event

```
property OnSeeked: TNotifyEvent
```

This event is triggered whenever the Seeking property reverts to False.

TAudio.OnSeeking Event

```
property OnSeeking: TNotifyEvent
```

This event is triggered whenever the playback location of the media control has changed for the current media. This is caused by the user requesting such an action via the user interface when the ShowControls property is True, or when the application modifies the CurrentTime property.

TAudio.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TAudio.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TAudio.OnStalled Event

```
property OnStalled: TNotifyEvent
```

This event is triggered whenever the media control is trying to load data for the current media, but no data is arriving over the network.

TAudio.OnSuspend Event

```
property OnSuspend: TNotifyEvent
```

This event is triggered whenever the media control has loaded enough data to enable playback, and has stopped loading more data.

TAudio.OnTimeUpdate Event

```
property OnTimeUpdate: TNotifyEvent
```

This event is triggered whenever the CurrentTime property changes.

Note
This event can be fired as many as 60 times per second in some web browsers, so be very careful about how time-consuming any event handlers are for this event.

TAudio.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TAudio.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TAudio.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TAudio.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

TAudio.OnVolumeChange Event

```
property OnVolumeChange: TNotifyEvent
```

This event is triggered whenever the audio volume of the media control has changed for the current media. This is caused by the user requesting such an action via the user interface when the ShowControls property is True, or when the application modifies the Volume property.

TAudio.OnWaiting Event

```
property OnWaiting: TNotifyEvent
```

This event is triggered whenever the media control cannot start/resume playback because more data needs to be loaded for the current media.

10.10 TAudioElement Component

Unit: WebUI

Inherits From TMediaElement

The TAudioElement class is the element class for audio UI elements, and contains all of the audio playback functionality in the form of public methods and properties/events that control classes can use to create audio controls.

> **Note**
> This element does not provide support for audio playback at design-time, and the applicable playback methods and properties are all stubs.

Properties	Methods	Events

10.11 TAutoSize Component

Unit: WebUI

Inherits From TElementAttribute

The TAutoSize class represents the auto-sizing attributes to use for the content of a UI element or control. The height, width, or both can be set as auto-sized. The content of a UI element or control that is used to determine the height and/or width can be text or HTML content, as well as the space consumed by child UI elements or controls.

Note
Not all controls support auto-sizing.

Properties	Methods	Events
Height	SetToDefault	
Width		

TAutoSize.Height Property

```
property Height: Boolean
```

Specifies that the height of the UI element or control should be automatically set based upon the height of its content and/or the height of any child UI elements or controls.

TAutoSize.Width Property

```
property Width: Boolean
```

Specifies that the width of the UI element or control should be automatically set based upon the width of its content and/or the width of any child UI elements or controls.

TAutoSize.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset all auto-size properties to their default values.

10.12 TBackground Component

Unit: WebUI

Inherits From TElementAttribute

The TBackground class represents the background of a UI element or control. Backgrounds can be solid colors (including transparent) or gradients, and can have tiled and non-tiled background images.

Properties	Methods	Events
Clip	SetToDefault	
Fill		
Image		
Origin		

TBackground.Clip Property

```
property Clip: TBackgroundOrientationType
```

Specifies how the background should be clipped within the UI element or control.

TBackground.Fill Property

```
property Fill: TFill
```

Specifies the background fill.

TBackground.Image Property

```
property Image: TBackgroundImage
```

Specifies the background image.

TBackground.Origin Property

```
property Origin: TBackgroundOrientationType
```

Specifies the origin of the background within the UI element or control.

TBackground.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the background's properties to their default values.

10.13 TBackgroundImage Component

Unit: WebUI

Inherits From TElementAttribute

The TBackgroundImage class represents the background image of a UI element or control. The background image is specified using the Name property and/or the Data (for base64-encoded, data-URI inline images).

Note
Any background images specified at design-time are automatically embedded in form and control interface files, are emitted as part of the application during compilation, and are automatically loaded during application initialization.

Properties	Methods	Events
Data	BeginAnimation	
Height	CancelAnimation	
Left	SetToDefault	
Name		
PositionType		
RepeatStyle		
SizeType		
Top		
Width		

TBackgroundImage.Data Property

```
property Data: String
```

Specifies the image as a base64-encoded, data-URI inline image. The image data should **not** include any data-URI prefixes. The interface manager automatically adds the proper prefixes when applying the background to the applicable UI element or control.

Note

If the Name property **and** the Data property are both assigned non-blank values, then the Name property is effectively ignored. Also, if the Name property is assigned a new value, the Data property will automatically be cleared.

TBackgroundImage.Height Property

```
property Height: Integer
```

Specifies the height of the background image. If the actual image height of the background image is different than this value, then the background image height is stretched/contracted accordingly.

Note
This property is only valid when the SizeType is stSpecified.

TBackgroundImage.Left Property

```
property Left: Integer
```

Specifies the left position of the background image.

Note
This property is only valid when the PositionType is ptSpecified.

TBackgroundImage.Name Property

```
property Name: String
```

Specifies the local image name, at design-time, or the URL of an image resource, at run-time.

> **Note**
> If the Name property **and** the Data property are both assigned non-blank values, then the Name property is effectively ignored. Also, if the Name property is assigned a new value, the Data property will automatically be cleared.

TBackgroundImage.PositionType Property

```
property PositionType: TBackgroundImagePositionType
```

Specifies how the background image should be positioned within the UI element or control.

Note
The bounding rectangle used for determining the positioning is based upon the TBackground Origin and Clip properties.

TBackgroundImage.RepeatStyle Property

```
property RepeatStyle: TBackgroundImageRepeatStyle
```

Specifies how the background image should be tiled within the UI element or control.

Note
The bounding rectangle used for determining the positioning is based upon the TBackground Origin and Clip properties.

TBackgroundImage.SizeType Property

```
property SizeType: TBackgroundImageSizeType
```

Specifies how the background image should be sized within the UI element or control.

Note
The bounding rectangle used for determining the positioning is based upon the TBackground Origin and Clip properties.

TBackgroundImage.Top Property

```
property Top: Integer
```

Specifies the top position of the background image.

Note
This property is only valid when the PositionType is ptSpecified.

TBackgroundImage.Width Property

```
property Width: Integer
```

Specifies the width of the background image. If the actual image width of the background image is different than this value, then the background image width is stretched/contracted accordingly.

Note
This property is only valid when the SizeType is stSpecified.

TBackgroundImage.BeginAnimation Method

```
procedure BeginAnimation(ADirection:
      TBackgroundImageAnimateDirection)
```

Use this method to begin animating the collection of animation frames in a background image.

TBackgroundImage.CancelAnimation Method

```
procedure CancelAnimation
```

Use this method to stop animating the collection of animation frames in a background image.

TBackgroundImage.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the background image's properties to their default values.

10.14 TBalloonLabel Component

Unit: WebLabels

Inherits From TBalloonLabelControl

The TBalloonLabel component represents a label control that looks like a word balloon, with customizations such as the orientation of the balloon tail and the ability to specify an icon along with the text.

Properties	Methods	Events
AutoHideTime		OnAnimationComplete
Background		OnAnimationsComplete
Caption		OnClick
Corners		OnDblClick
Cursor		OnHide
DataColumn		OnMouseDown
DataSet		OnMouseEnter
Font		OnMouseLeave
Format		OnMouseMove
Hint		OnMouseUp
Icon		OnMove
Opacity		OnShow
Orientation		OnSize
Padding		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TBalloonLabel.AutoHideTime Property

```
property AutoHideTime: Integer
```

Specifies the number of milliseconds that the balloon label control should be shown before automatically hiding itself. The default value is 0 milliseconds, which prevents the control from automatically hiding itself.

TBalloonLabel.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TBalloonLabel.Caption Property

```
property Caption: TCaption
```

Specifies the text to be shown in the balloon label control. The text can contain line feeds. The default value is ''.

TBalloonLabel.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TBalloonLabel.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TBalloonLabel.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TBalloonLabel.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TBalloonLabel.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TBalloonLabel.Format Property

```
property Format: TFormat
```

Specifies the content formatting to use for the control's Caption.

TBalloonLabel.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TBalloonLabel.Icon Property

```
property Icon: TIconProperties
```

Specifies the properties of the icon used with the control.

TBalloonLabel.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TBalloonLabel.Orientation Property

```
property Orientation: TBalloonOrientation
```

Specifies the orientation of the tail for the balloon label.

TBalloonLabel.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

TBalloonLabel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TBalloonLabel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TBalloonLabel.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TBalloonLabel.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TBalloonLabel.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TBalloonLabel.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TBalloonLabel.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TBalloonLabel.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TBalloonLabel.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TBalloonLabel.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TBalloonLabel.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TBalloonLabel.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TBalloonLabel.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TBalloonLabel.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TBalloonLabel.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TBalloonLabel.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TBalloonLabel.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.15 TBalloonLabelControl Component

Unit: WebLabels

Inherits From TBindableColumnControl

The TBalloonLabelControl control is the base class for balloon label controls, and contains all of the label functionality in the form of public methods and protected properties/events that descendant classes can use to create customized balloon label controls.

Properties	Methods	Events

10.16 TBasicPanel Component

Unit: WebCtnrs

Inherits From TBasicPanelControl

The TBasicPanel component represents a basic panel control that cannot be scrolled.

Properties	Methods	Events
ActivateOnClick		OnAnimationComplete
AutoSize		OnAnimationsComplete
Background		OnClick
Border		OnDblClick
Corners		OnHide
Cursor		OnKeyDown
Hint		OnKeyPress
InsetShadow		OnKeyUp
Opacity		OnMouseDown
OutsetShadow		OnMouseEnter
Padding		OnMouseLeave
TabOrder		OnMouseMove
TabStop		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TBasicPanel.ActivateOnClick Property

```
property ActivateOnClick: Boolean
```

Specifies whether the control should automatically be brought to the front when it, or any child controls, are clicked.

TBasicPanel.AutoSize Property

```
property AutoSize: TAutoSize
```

Specifies how (if at all) the control should automatically be sized based upon the child controls placed in the panel.

TBasicPanel.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TBasicPanel.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TBasicPanel.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TBasicPanel.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TBasicPanel.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TBasicPanel.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Specifies the inset shadow for the control.

TBasicPanel.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TBasicPanel.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the control.

TBasicPanel.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

TBasicPanel.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TBasicPanel.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TBasicPanel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TBasicPanel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TBasicPanel.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TBasicPanel.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TBasicPanel.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TBasicPanel.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when a child control has input focus and the user presses a key or key combination.

TBasicPanel.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when a child control has input focus and presses/releases a key or key combination.

TBasicPanel.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when a child control has input focus and the user releases a key or key combination.

TBasicPanel.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TBasicPanel.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TBasicPanel.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TBasicPanel.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TBasicPanel.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TBasicPanel.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TBasicPanel.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TBasicPanel.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TBasicPanel.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TBasicPanel.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TBasicPanel.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TBasicPanel.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.17 TBasicPanelControl Component

Unit: WebCtnrs

Inherits From TControl

The TBasicPanelControl control is the base class for basic panel controls, and contains all of the panel functionality in the form of public methods and protected properties/events that descendant classes can use to create customized panel controls.

Properties	Methods	Events

10.18 TBindableColumnControl Component

Unit: WebCtrls

Inherits From TBindableControl

The TBindableColumnControl control is the base class for controls that bind to dataset columns, and contains all of the dataset column binding functionality in the form of public methods and protected properties/events that descendant classes can use to create customized controls that bind to dataset columns.

Properties	Methods	Events

10.19 TBindableControl Component

Unit: WebCtrls

Inherits From TControl

The TBindableControl control is the base class for controls that bind to datasets. It contains all of the dataset binding functionality in the form of public methods and protected properties/events that descendant classes can use to create customized controls that bind to datasets.

Properties	Methods	Events

10.20 TBlobValue Component

Unit: WebCore

Inherits From TStringValue

This class represents the value for a BLOB (Binary Large Object) column in a row in a TDataSet component.

> **Note**
> BLOB columns use string values to represent URLs from where the actual binary resources can be retrieved.

Properties	Methods	Events

10.21 TBodyElement Component

Unit: WebUI

Inherits From TElement

The TBodyElement class is the element class used as the root element by the interface manager at run-time. It wraps the browser document's body element, and a visual application's Surface uses this element as its base element.

Properties	Methods	Events

10.22 TBooleanValue Component

Unit: WebCore

Inherits From TDataValue

This class represents the value for a Boolean column in a row in a TDataSet component.

Properties	Methods	Events

10.23 TBorder Component

Unit: WebUI

Inherits From TElementAttribute

The TBorder class represents the border for a UI element or control.

Properties	Methods	Events
Bottom	SetToDefault	
Left		
Right		
Top		

TBorder.Bottom Property

```
property Bottom: TBorderSide
```

Specifies the bottom border side.

TBorder.Left Property

```
property Left: TBorderSide
```

Specifies the left border side.

TBorder.Right Property

```
property Right: TBorderSide
```

Specifies the right border side.

TBorder.Top Property

```
property Top: TBorderSide
```

Specifies the top border side.

TBorder.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the border's properties to their default values.

10.24 TBorderSide Component

Unit: WebUI

Inherits From TElementAttribute

The TBorderSide class represents one side of the TBorder for a UI element or control.

Properties	Methods	Events
Color	SetToDefault	
Offset		
Style		
Visible		
Width		

TBorderSide.Color Property

```
property Color: TColor
```

Specifies the color of the border side.

TBorderSide.Offset Property

```
property Offset: Integer
```

Indicates the actual offset, in pixels, of the border. At run-time in a browser, a UI element or control's border side Width property is only taken into account when the border side is visible.

TBorderSide.Style Property

```
property Style: TBorderStyle
```

Specifies the style of the border side.

TBorderSide.Visible Property

```
property Visible: Boolean
```

Specifies whether the border side is visible.

TBorderSide.Width Property

```
property Width: Integer
```

Specifies the width, in pixels, of the border side.

TBorderSide.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the border side's properties to their default values.

10.25 TBoundingAttribute Component

Unit: WebUI

Inherits From TElementAttribute

The TBoundingAttribute class represents a series of left, top, right, and bottom bounding values, in pixels, for a UI element or control. It is the base class for the TMargins and TPadding classes.

Properties	Methods	Events
Bottom	SetToDefault	
Left		
Right		
Top		

TBoundingAttribute.Bottom Property

```
property Bottom: Integer
```

Specifies the bottom bounding value.

TBoundingAttribute.Left Property

```
property Left: Integer
```

Specifies the left bounding value.

TBoundingAttribute.Right Property

```
property Right: Integer
```

Specifies the right bounding value.

TBoundingAttribute.Top Property

```
property Top: Integer
```

Specifies the top bounding value.

TBoundingAttribute.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the bounding attribute properties to their default values.

10.26 TBrowser Component

Unit: WebBrwsr

Inherits From TWebControl

The TBrowser component represents an HTML document container control, and can be used as the output destination for HTML Forms via the THTMLForm Output property.

Properties	Methods	Events
Background	Print	OnAnimationComplete
Border		OnAnimationsComplete
Corners		OnHide
Cursor		OnLoad
DataColumn		OnMove
DataSet		OnShow
Document		OnSize
DocumentText		OnUnload
Hint		
InsetShadow		
Loaded		
Opacity		
OutsetShadow		
Padding		
Scrolling		
URL		

TBrowser.Background Property

```
property Background: TBackground
```

Specifies the background for the control.

TBrowser.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TBrowser.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TBrowser.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TBrowser.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TBrowser.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TBrowser.Document Property

```
property Document: THTMLDocument
```

Returns the DOM (Document Object Model) document instance of the currently-loaded HTML document. If the URL property has been specified, then this property will return the document instance once the OnLoad event has been triggered and the Loaded property is True.

Note

Accessing the DOM document instance allows you to manipulate the children of the DOM document instance in code instead of having to use HTML strings, which is the case when using the DocumentText property. However, this access is subject to same-origin security constraints, and will be denied if the contents of the TBrowser instance were loaded from a different origin.

TBrowser.DocumentText Property

```
property DocumentText: String
```

Returns the currently-loaded HTML document in the control as a string. If the URL property has been specified, then this property will return the document contents once the OnLoad event has been triggered and the Loaded property is True.

Note
 You can also assign a valid HTML string to this property, in which case the URL property is automatically cleared.

TBrowser.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TBrowser.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Specifies the inset shadow for the control.

TBrowser.Loaded Property

```
property Loaded: Boolean
```

Indicates whether the HTML document specified by the URL property has been loaded.

An event handler can be attached to the OnLoad event to execute code when the document is loaded.

TBrowser.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TBrowser.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the control.

TBrowser.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

TBrowser.Scrolling Property

```
property Scrolling: Boolean
```

Specifies whether scrolling should be enabled for the control.

Note
 This property is required because certain browsers require a special way of specifying whether scrollbars should be shown for embedded browser controls.

TBrowser.URL Property

```
property URL: String
```

Specifies the URL for the HTML document. Whenever the URL property changes, the OnUnload event is triggered immediately. The OnLoad event is triggered once the document has been loaded.

TBrowser.Print Method

```
procedure Print
```

Use this method to print the currently-loaded HTML document in the control. If no HTML document is loaded, then this method does nothing.

TBrowser.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TBrowser.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TBrowser.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TBrowser.OnLoad Event

```
property OnLoad: TNotifyEvent
```

This event is triggered when the HTML document specified by the URL property has been completely loaded.

TBrowser.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TBrowser.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TBrowser.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TBrowser.OnUnload Event

```
property OnUnload: TNotifyEvent
```

This event is triggered when the currently-loaded HTML document specified by the URL or DocumentText property has been unloaded.

10.27 TButton Component

Unit: WebBtns

Inherits From TButtonControl

The TButton component represents a button control. A button control allows the user to initiate a specific action by using a mouse click or by pushing the spacebar or enter key.

Properties	Methods	Events
AutoWidth		OnAnimationComplete
Caption		OnAnimationsComplete
Corners		OnClick
Cursor		OnEnter
Enabled		OnExit
Font		OnHide
Hint		OnKeyDown
Icon		OnKeyPress
TabOrder		OnKeyUp
TabStop		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TButton.AutoWidth Property

```
property AutoWidth: Boolean
```

Specifies whether the width of the button should be automatically set based upon the Caption, Icon, and Font properties.

TButton.Caption Property

```
property Caption: String
```

Specifies the textual caption to display on the button control. The default value is ''.

TButton.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TButton.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TButton.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TButton.Font Property

```
property Font: TFont
```

Specifies the font to use for the caption on the button control.

TButton.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TButton.Icon Property

```
property Icon: TIconProperties
```

Specifies the properties of the icon used with the control.

TButton.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TButton.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TButton.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TButton.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TButton.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TButton.OnEnter Event

```
property OnEnter: TNotifyEvent
```

This event is triggered when the control obtains input focus.

TButton.OnExit Event

```
property OnExit: TNotifyEvent
```

This event is triggered when the control loses input focus.

TButton.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TButton.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when the control has input focus and the user presses a key or key combination.

TButton.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TButton.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when the control has input focus and the user releases a key or key combination.

TButton.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TButton.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TButton.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TButton.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TButton.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TButton.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TButton.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TButton.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TButton.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TButton.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TButton.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TButton.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.28 TButtonComboBox Component

Unit: WebEdits

Inherits From TButtonComboControl

The TButtonComboBox component represents a button combo box control. A button combo box is a combo control that allows the user to select an input value from a drop-down list of values by using a mouse click or by pushing the spacebar or enter key.

> **Note**
> This type of control does not allow for direct editing of the input value, and is ideal for touch environments where you may not want a soft keyboard to appear when such a control gains focus.

Properties	Methods	Events
Alignment		OnAnimationComplete
AutoDropDown		OnAnimationsComplete
AutoItemHeight		OnChange
Cursor		OnClick
DataColumn		OnDropDownHide
DataSet		OnDropDownShow
Direction		OnEnter
DropDownItemCount		OnExit
DropDownPosition		OnHide
Enabled		OnKeyDown
Font		OnKeyPress
Hint		OnKeyUp
ItemHeight		OnMouseDown
ItemIndex		OnMouseEnter
Items		OnMouseLeave
KeyPressInterval		OnMouseMove
ReadOnly		OnMouseUp
Sorted		OnMove
TabOrder		OnShow
TabStop		OnSize
Text		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TButtonComboBox.Alignment Property

```
property Alignment: TContentAlignment
```

Specifies the alignment of the input value for the control.

TButtonComboBox.AutoDropDown Property

```
property AutoDropDown: Boolean
```

Specifies that the drop-down list of Items should automatically be shown when the user starts typing. The default value is False.

TButtonComboBox.AutoItemHeight Property

```
property AutoItemHeight: Boolean
```

Specifies that the displayed height of the drop-down items will automatically be set based upon the Font property settings. The default value is True.

TButtonComboBox.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TButtonComboBox.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TButtonComboBox.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TButtonComboBox.Direction Property

```
property Direction: TContentDirection
```

Specifies the direction in which the caption is displayed.

TButtonComboBox.DropDownItemCount Property

```
property DropDownItemCount: Integer
```

Specifies the number of visible items to display in the drop-down list of Items.

TButtonComboBox.DropDownPosition Property

```
property DropDownPosition: TDropDownPosition
```

Specifies where the drop-down list will be positioned.

TButtonComboBox.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TButtonComboBox.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TButtonComboBox.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TButtonComboBox.ItemHeight Property

```
property ItemHeight: Integer
```

Specifies the height, in pixels, of the items displayed in the drop-down list.

TButtonComboBox.ItemIndex Property

```
property ItemIndex: Integer
```

Specifies the index of the selected item in the drop-down list of Items, or -1 if there is no selected item.

TButtonComboBox.Items Property

```
property Items: TStrings
```

Specifies the items to use for the drop-down list.

TButtonComboBox.KeyPressInterval Property

```
property KeyPressInterval: Integer
```

Specifies the interval, in milliseconds, that is used by the control to combine user keystrokes into a search value that is then used for performing a near search on the Items property. Effectively, this means that the user has KeyPressInterval milliseconds in which to hit a key in order for the keystroke to be included as part of a near search. The default value is 300 milliseconds.

For example, if the user hits the 'S', 'M', and 'I' keys within the KeyPressInterval property value, but hits the 'T' key outside of the KeyPressInterval property, then the control will perform a near search using the value 'SMI', followed by a near search using the value 'T'.

TButtonComboBox.ReadOnly Property

```
property ReadOnly: Boolean
```

Specifies whether the control's input value can be modified by the user. The default value is False.

Note
The input value can always be programmatically modified.

TButtonComboBox.Sorted Property

```
property Sorted: Boolean
```

Specifies whether the drop-down items will automatically be sorted. The default value is False.

TButtonComboBox.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TButtonComboBox.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TButtonComboBox.Text Property

```
property Text: String
```

Specifies the control's input value as a string.

TButtonComboBox.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TButtonComboBox.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TButtonComboBox.OnChange Event

```
property OnChange: TNotifyEvent
```

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TButtonComboBox.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TButtonComboBox.OnDropDownHide Event

```
property OnDropDownHide: TNotifyEvent
```

This event is triggered when the associated drop-down control is hidden.

TButtonComboBox.OnDropDownShow Event

```
property OnDropDownShow: TNotifyEvent
```

This event is triggered when the associated drop-down control is shown.

TButtonComboBox.OnEnter Event

```
property OnEnter: TNotifyEvent
```

This event is triggered when the control obtains input focus.

TButtonComboBox.OnExit Event

```
property OnExit: TNotifyEvent
```

This event is triggered when the control loses input focus.

TButtonComboBox.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TButtonComboBox.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when the control has input focus and the user presses a key or key combination.

TButtonComboBox.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TButtonComboBox.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when the control has input focus and the user releases a key or key combination.

TButtonComboBox.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TButtonComboBox.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TButtonComboBox.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TButtonComboBox.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TButtonComboBox.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TButtonComboBox.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TButtonComboBox.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TButtonComboBox.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TButtonComboBox.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TButtonComboBox.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TButtonComboBox.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TButtonComboBox.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.29 TButtonComboControl Component

Unit: WebEdits

Inherits From TDropDownButtonControl

The TButtonComboControl control is the base class for button combo controls, and contains all of the button combo functionality in the form of public methods and protected properties/events that descendant classes can use to create customized button combo controls.

Properties	Methods	Events

10.30 TButtonControl Component

Unit: WebBtns

Inherits From TControl

The TButtonControl control is the base class for button controls, and contains all of the core button functionality in the form of public methods and protected properties/events that descendant classes can use to create customized button controls.

Properties	Methods	Events

10.31 TButtonInputControl Component

Unit: WebEdits

Inherits From TInputControl

The TButtonInputControl control is the base class for button-style input controls, and contains all of the core input functionality in the form of public methods and protected properties/events that descendant classes can use to create customized button-style input controls.

Properties	Methods	Events

10.32 TCalendar Component

Unit: WebCals

Inherits From TCalendarControl

The TCalendar component represents a calendar control for selecting a date. The user can change the active period by pressing the page up/page down keys, and can change the calendar View property by pressing the numeric keypad plus and minus keys.

Properties	Methods	Events
Background		OnAnimationComplete
Border		OnAnimationsComplete
Corners		OnChange
Cursor		OnClick
DataColumn		OnDblClick
DataSet		OnHide
DefaultView		OnKeyDown
Enabled		OnKeyPress
Font		OnKeyUp
Hint		OnMouseDown
LocalizeText		OnMouseEnter
ReadOnly		OnMouseLeave
TabOrder		OnMouseMove
TabStop		OnMouseUp
Text		OnMouseWheel
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TCalendar.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TCalendar.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TCalendar.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TCalendar.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TCalendar.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TCalendar.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TCalendar.DefaultView Property

```
property DefaultView: TCalendarView
```

Specifies the default view for the calendar control. The default view determines both the initial view shown in the calendar after it is created, as well as the minimum view that the user is permitted to navigate to. The default value is cvMonth.

TCalendar.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TCalendar.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TCalendar.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TCalendar.LocalizeText Property

```
property LocalizeText: Boolean
```

Specifies whether date assignments (as strings) to the control's Text property are treated as local dates or UTC dates. The default value is True.

Note
This property only affects the value of the SelectedDate property and does not affect how date values are saved to and from a dataset column via the DataSet and DataColumn properties.

TCalendar.ReadOnly Property

```
property ReadOnly: Boolean
```

Specifies whether the control's input value can be modified by the user. The default value is False.

Note
The input value can always be programmatically modified.

TCalendar.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TCalendar.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TCalendar.Text Property

```
property Text: String
```

Specifies the control's input value as a string.

TCalendar.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TCalendar.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TCalendar.OnChange Event

```
property OnChange: TNotifyEvent
```

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TCalendar.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TCalendar.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TCalendar.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TCalendar.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when the control has input focus and the user presses a key or key combination.

TCalendar.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TCalendar.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when the control has input focus and the user releases a key or key combination.

TCalendar.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TCalendar.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TCalendar.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TCalendar.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TCalendar.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TCalendar.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

This event is triggered whenever the mouse wheel is rotated forward or backward.

TCalendar.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TCalendar.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TCalendar.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TCalendar.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TCalendar.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TCalendar.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TCalendar.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.33 TCalendarControl Component

Unit: WebCals

Inherits From TInputControl

The TCalendarControl control is the base class for calendar controls, and contains all of the core calendar functionality in the form of public methods and protected properties/events that descendant classes can use to create customized calendar controls.

Properties	Methods	Events
SelectedDate	NextPeriod	
View	PriorPeriod	

TCalendarControl.SelectedDate Property

```
property SelectedDate: DateTime
```

Specifies the selected date displayed in the calendar.

TCalendarControl.View Property

```
property View: TCalendarView
```

Specifies the active calendar view.

TCalendarControl.NextPeriod Method

```
procedure NextPeriod
```

Use this method to navigate to the next period in the calendar control. The next period displayed is determined by the active view.

TCalendarControl.PriorPeriod Method

```
procedure PriorPeriod
```

Use this method to navigate to the prior period in the calendar control. The prior period displayed is determined by the active view.

10.34 TCanvasElement Component

Unit: WebUI

Inherits From TElement

The TCanvasElement class is the element class for HTML5 canvas elements, and contains all of the canvas functionality in the form of public methods and properties/events that control classes can use to create painting/drawing controls.

> **Note**
> This element does not provide support for canvas drawing at design-time, and the applicable drawing methods and properties are all stubs.

Properties	Methods	Events
Alpha	Arc	
CompositeOperation	ArcTo	
FillColor	BeginPath	
FillGradient	BezierCurveTo	
FillPattern	ClearRect	
FillStyle	Clip	
LineCapStyle	ClosePath	
LineJoinStyle	ConvertToDataURL	
LineWidth	DrawImage	
MiterLimit	Fill	
ShadowBlur	FillRect	
ShadowColor	FillText	
ShadowOffsetX	IsPointInPath	
ShadowOffsetY	LineTo	
StrokeColor	MeasureText	
StrokeGradient	MoveTo	
StrokePattern	QuadraticCurveTo	
StrokeStyle	Rect	
TextAlign	Rotate	
TextBaseLine	Save	
	Scale	
	SetTransform	
	Stroke	
	StrokeRect	
	StrokeText	
	Transform	
	Translate	

TCanvasElement.Alpha Property

```
property Alpha: Double
```

Specifies the global alpha value (transparency) of the canvas. This value is multiplied by the alpha value (transparency) of all drawn pixels. The default is 1.0, meaning that the transparency of the drawn pixels is not modified, and the value must be between 0.0 and 1.0.

TCanvasElement.CompositeOperation Property

```
property CompositeOperation: TCompositeOperation
```

Specifies how newly-drawn pixels (source) are combined with the existing pixels on the canvas (destination) during drawing operations. The default value is coSourceOver.

TCanvasElement.FillColor Property

```
property FillColor: TColor
```

Specifies the color to use with fill operations when the FillStyle property is set to dsColor.

TCanvasElement.FillGradient Property

```
property FillGradient: TCanvasGradient
```

Specifies the gradient to use with fill operations when the FillStyle property is set to dsGradient.

TCanvasElement.FillPattern Property

```
property FillPattern: TCanvasPattern
```

Specifies the pattern to use with fill operations when the FillStyle property is set to dsPattern.

TCanvasElement.FillStyle Property

```
property FillStyle: TDrawStyle
```

Specifies how fill operations will draw on the canvas.

TCanvasElement.LineCapStyle Property

```
property LineCapStyle: TLineCapStyle
```

Specifies how lines should be terminated when drawing wide lines on the canvas. The default value is csButt.

TCanvasElement.LineJoinStyle Property

```
property LineJoinStyle: TLineJoinStyle
```

Specifies how wide lines should be drawn when they intersect on the canvas. The default value is jsMiter. The MiterLimit property is used to limit the length of a miter join.

TCanvasElement.LineWidth Property

```
property LineWidth: Double
```

Specifies the line width to use for line drawing (stroking) operations on the canvas.

TCanvasElement.MiterLimit Property

```
property MiterLimit: Double
```

Specifies the upper limit on miter joins when the LineJoinStyle is set to jsMiter. The default value is 10 (pixels).

TCanvasElement.ShadowBlur Property

```
property ShadowBlur: Double
```

Specifies how much blur shadows should have. The default value is 0, which produces shadows with a crisp edge.

TCanvasElement.ShadowColor Property

```
property ShadowColor: TColor
```

Specifies the color of the shadows. The default value is clBlack.

TCanvasElement.ShadowOffsetX Property

```
property ShadowOffsetX: Double
```

Specifies the horizontal offset of shadows. The default value is 0.

TCanvasElement.ShadowOffsetY Property

```
property ShadowOffsetY: Double
```

Specifies the vertical offset of shadows. The default value is 0.

TCanvasElement.StrokeColor Property

```
property StrokeColor: TColor
```

Specifies the color to use with stroke (line-drawing) operations when the StrokeStyle property is set to dsColor.

TCanvasElement.StrokeGradient Property

```
property StrokeGradient: TCanvasGradient
```

Specifies the gradient to use with stroke (line-drawing) operations when the StrokeStyle property is set to dsGradient.

TCanvasElement.StrokePattern Property

```
property StrokePattern: TCanvasPattern
```

Specifies the pattern to use with stroke (line-drawing) operations when the StrokeStyle property is set to dsPattern.

TCanvasElement.StrokeStyle Property

```
property StrokeStyle: TDrawStyle
```

Specifies how stroke (line-drawing) operations will draw on the canvas.

TCanvasElement.TextAlign Property

```
property TextAlign: TTextAlignment
```

Specifies how text is aligned when it is drawn on the canvas. The default value is taLeftJustify.

TCanvasElement.TextBaseLine Property

```
property TextBaseLine: TTextBaseLine
```

Specifies the vertical alignment of text when it is drawn on the canvas. The default value is blAlphabetic.

TCanvasElement.Arc Method

```
procedure Arc(X,Y: Double; Radius: Double; StartAngle, EndAngle:
      Double; CounterClockwise: Boolean=False)
```

Use this method to add an arc to the current subpath of the canvas. The first three parameters specify the center and radius of a circle. The next two parameters are angles (in radians) that specify the start and end points of an arc along the circle. The last parameter specifies whether the arc is traversed counterclockwise or clockwise along the circle's circumference.

TCanvasElement.ArcTo Method

```
procedure ArcTo(X1,Y1,X2,Y2: Double; Radius: Double)
```

Use this method to add a straight line and an arc to the current subpath of the canvas. The first two parameters specify a starting point and the second two parameters specify an ending point. The arc that is added to the subpath is a portion of a circle with the specified radius. The arc begins and ends at the two specified points, with a line first added to the subpath to connect the current canvas position to the starting point. After drawing the arc, the canvas position is set to the ending point.

TCanvasElement.BeginPath Method

```
procedure BeginPath
```

Use this method to discard any existing defined subpath for the canvas and begin a new subpath. After this method is called, there is no current point for the canvas.

Note
This method is implicitly called when the canvas is first created.

TCanvasElement.BezierCurveTo Method

```
procedure BezierCurveTo(CPX1,CPY1,CPX2,CPY2: Double; X,Y:
      Double)
```

Use this method to add a cubic Bezier curve to the current subpath of the canvas. The first four parameters define the shape of the curve. The starting point of the curve is the current point of the canvas and the end point is defined by the last two parameters. After this method is called, the current point on the canvas is the specified end point.

TCanvasElement.ClearRect Method

```
procedure ClearRect(X,Y: Double; Width,Height: Double)
```

Use this method to clear the specified rectangle. Clearing the rectangle is equivalent to filling it with a transparent black (clBlack) color.

Note
This method does not affect the current path or current point of the canvas.

TCanvasElement.Clip Method

```
procedure Clip
```

Use this method to compute the intersection of the inside of the current path with the current clipping region and use the smaller region as the new clipping region.

Note
Clipping regions cannot be enlarged, so if you only need a temporary clipping region, you should call the Save method to save the current clipping region, create the smaller clipping region, and then restore the previous clipping region by calling the Restore method.

TCanvasElement.ClosePath Method

```
procedure ClosePath
```

Use this method to close the current subpath of the canvas, if it is open. The subpath is closed by adding a line connecting the current point to the first point of the subpath. A new subpath is then started at the ending point of this last subpath.

Note
The Fill and Clip methods always treat the subpath as closed, so the only time you should need to call this method is when you want to Stroke a closed subpath.

TCanvasElement.ConvertToDataURL Method

```
function ConvertToDataURL(const ImageType: String; ImageQuality:
      Integer=100): String
```

Use this method to convert the contents of the canvas to an image represented as a Base64-encoded data URL string. Use the ImageType parameter to specify the image format via its MIME type, and the ImageQuality parameter to indicate the image quality for formats that support the specification of the image quality, such as the JPEG format.

Note
Currently, most web browsers only support 'image/png' or 'image/jpeg' as the ImageType parameter. Please consult the web browser documentation on whether additional formats are supported in a specific browser.

TCanvasElement.DrawImage Method

```
procedure DrawImage(Image: TElement; X,Y: Double)

procedure DrawImage(Image: TElement; X,Y: Double; Width,Height:
      Double)

procedure DrawImage(Image: TElement; SourceX,SourceY: Double;
      SourceWidth,SourceHeight: Double; DestX,DestY: Double;
      DestWidth,DestHeight: Double)
```

Use this method to draw an image, or a portion of an image, at a specified point or, optionally, into a specified rectangle.

The first version of this method simply draws the image at a specified point, retaining the image's original height and width.

The second version of this method draws the image at a specified point, but also scales the image so that it matches the specified height and width.

The last version of this method draws copies the specified rectangle from the image to the specified rectangle on the canvas.

Note
If the specified TElement instance is a TImageElement instance, then it must be completely loaded before passing it to this method. If the image element has not been completely loaded, then an exception will be raised. You can use the Loaded property to determine if an image element has been completely loaded yet.

TCanvasElement.Fill Method

```
procedure Fill
```

Use this method to fill the current path of the canvas with the color, gradient, or pattern specified by the FillStyle, FillColor, FillGradient, and FillPattern properties.

Note
If the current subpath is not closed, then this method will treat it as closed. However, it won't actually close the subpath.

TCanvasElement.FillRect Method

```
procedure FillRect(X,Y: Double; Width,Height: Double)
```

Use this method to fill the specified rectangle with the color, gradient, or pattern specified by the FillStyle, FillColor, FillGradient, and FillPattern properties.

Note
This method does not affect the current path or current point of the canvas.

TCanvasElement.FillText Method

```
procedure FillText(const Text: String; X,Y: Double; MaxWidth:
      Double=-1)
```

Use this method to draw the specified text at the specified point with the font specified by the Font property, and the color, gradient, or pattern specified by the FillStyle, FillColor, FillGradient, and FillPattern properties.

The optional MaxWidth parameter specifies a maximum width for the text. If the width of the text exceeds this value, then the text is drawn using a condensed font so that it fits within the specified width.

TCanvasElement.IsPointInPath Method

```
function IsPointInPath(X,Y: Double): Boolean
```

Use this method to determine if the specified point falls within or on the edge of the current path of the canvas.

TCanvasElement.LineTo Method

```
procedure LineTo(X,Y: Double)
```

Use this method to add a straight line to the current subpath of the canvas, with the current point used as the starting point of the line. After this method is called, the current point of the canvas is set to the specified ending point.

TCanvasElement.MeasureText Method

```
function MeasureText(const Text: String): Double
```

Use this method to measure the width of the specified text as it would be drawn using the Font property.

TCanvasElement.MoveTo Method

```
procedure MoveTo(X,Y: Double)
```

Use this method to set the current point of the canvas to the specified point and begin a new subpath at the same point.

Note
If the previous subpath only consisted of one point, then it is discarded when this method is called.

TCanvasElement.QuadraticCurveTo Method

```
procedure QuadraticCurveTo(CPX,CPY: Double; X,Y: Double)
```

Use this method to add a quadratic Bezier curve segment to the current subpath of the canvas. The curve starts at the current point of the canvas and ends at the point specified by the last two parameters. The first two parameters specify the shape of the curve between these two points. After this method is called, the current point on the canvas is the specified end point.

TCanvasElement.Rect Method

```
procedure Rect(X,Y: Double; Width, Height: Double)
```

Use this method to add a rectangle to the current path of the canvas. The added rectangle has its own subpath that is not connected to any other subpaths in the current path. After this method is called, the current point on the canvas is set to the point specified by the first two parameters.

TCanvasElement.Rotate Method

```
procedure Rotate(Angle: Double)
```

Use this method to change the transformation matrix of the canvas so that any subsequent drawing appears rotated by the specified angle (in radians).

TCanvasElement.Save Method

```
procedure Save
```

Use this method to push the current canvas properties on to the stack of saved states for the canvas. You can use the Restore method to pop the last saved state from the stack.

TCanvasElement.Scale Method

```
procedure Scale(SX,SY: Double)
```

Use this method to change the transformation matrix so that the scale of the canvas is modified according to the specified x and y axis values (independent of one another).

Note
Specify negative values to "flip" the coordinates of the canvas for a given axis.

TCanvasElement.SetTransform Method

```
procedure SetTransform(M11,M12,M21,M22: Double; DX,DY: Double)
```

Use this method to directly set the current transformation matrix for the canvas.

TCanvasElement.Stroke Method

```
procedure Stroke
```

Use this method to stroke the current path of the canvas with the color, gradient, or pattern specified by the StrokeStyle, StrokeColor, StrokeGradient, or StrokePattern properties. The LineCapStyle, LineJoinStyle, MiterLimit and LineWidth properties control how the lines that make up the current path are drawn.

TCanvasElement.StrokeRect Method

```
procedure StrokeRect(X,Y: Double; Width,Height: Double)
```

Use this method to stroke the specified rectangle with the color, gradient, or pattern specified by the StrokeStyle, StrokeColor, StrokeGradient, or StrokePattern properties. The LineCapStyle, LineJoinStyle, MiterLimit and LineWidth properties control how the lines that make up the rectangle are drawn.

Note
This method does not affect the current path or current point of the canvas.

TCanvasElement.StrokeText Method

```
procedure StrokeText(const Text: String; X,Y: Double; MaxWidth:
      Double=-1)
```

Use this method draw the outline of the specified text at the specified point with the font specified by the Font property, and the color, gradient, or pattern specified by the StrokeStyle, StrokeColor, StrokeGradient, or StrokePattern properties. The LineCapStyle, LineJoinStyle, MiterLimit and LineWidth properties control how the lines that make up the outline of the font characters are drawn.

The optional MaxWidth parameter specifies a maximum width for the text. If the width of the text exceeds this value, then the text is drawn using a condensed font so that it fits within the specified width.

TCanvasElement.Transform Method

```
procedure Transform(M11,M12,M21,M22: Double; DX,DY: Double)
```

Use this method to directly modify the existing transformation matrix for the canvas.

TCanvasElement.Translate Method

```
procedure Translate(DX,DY: Double)
```

Use this method to add horizontal and vertical offsets to the existing transformation matrix for the canvas.

10.35 TCanvasGradient Component

Unit: WebUI

Inherits From TObject

The TCanvasGradient class represents a linear or radial gradient. It is used by the TCanvasElement class for filling and stroking operations on a canvas.

Properties	Methods	Events
EndRadius	AddColorStop	
EndX	Create	
EndY	RemoveColorStops	
GradientType		
StartRadius		
StartX		
StartY		

TCanvasGradient.EndRadius Property

```
property EndRadius: Double
```

Specifies the ending radius for the gradient.

Note
This property only applies to radial gradients (GradientType of gtRadial);

TCanvasGradient.EndX Property

```
property EndX: Double
```

Specifies the ending x axis point for the gradient.

TCanvasGradient.EndY Property

```
property EndY: Double
```

Specifies the ending y axis point for the gradient.

TCanvasGradient.GradientType Property

```
property GradientType: TGradientType
```

Specifies the type of gradient, linear or radial, that is required for the fill or stroke operations on the canvas. The default gradient type is gtLinear.

TCanvasGradient.StartRadius Property

```
property StartRadius: Double
```

Specifies the starting radius for the gradient.

Note
This property only applies to radial gradients (GradientType of gtRadial);

TCanvasGradient.StartX Property

```
property StartX: Double
```

Specifies the starting x axis point for the gradient.

TCanvasGradient.StartY Property

```
property StartY: Double
```

Specifies the starting y axis point for the gradient.

TCanvasGradient.AddColorStop Method

```
procedure AddColorStop(Position: Double; Color: TColor)
```

Use this method to add a color stop to the gradient. Color stops are specified as a position between 0.0 and 1.0 along with a color that will start at that position. After specifying two or more color stops, the gradient will smoothly interpolate colors between the various stops.

Note
If you only specify a single color stop, then the gradient will display that color as a solid color with no interpolation.

TCanvasGradient.Create Method

```
constructor Create(ACanvas: TCanvasElement)
```

Use this method to create a new instance of the TCanvasGradient class. The ACanvasElement parameter indicates the canvas element that will contain the instance.

TCanvasGradient.RemoveColorStops Method

```
procedure RemoveColorStops
```

Use this method to remove all color stops for the gradient.

10.36 TCanvasPattern Component

Unit: WebUI

Inherits From TObject

The TCanvasPattern class represents a pattern comprised of a single image or an image that is tiled horizontally, vertically, or both. It is used by the TCanvasElement class for filling and stroking operations on a canvas.

Properties	Methods	Events
Image	Create	
RepeatStyle		

TCanvasPattern.Image Property

```
property Image: TElement
```

Specifies the image to use as the pattern for the fill or stroke operations on a canvas.

TCanvasPattern.RepeatStyle Property

```
property RepeatStyle: TPatternRepeatStyle
```

Specifies how the image specified by the Image property should be tiled when the pattern is used for filling and stroking operations.

TCanvasPattern.Create Method

```
constructor Create(ACanvas: TCanvasElement)
```

Use this method to create a new instance of the TCanvasPattern class. The ACanvasElement parameter indicates the canvas element that will contain the instance.

10.37 TCanvasPoint Component

Unit: WebUI

Inherits From TObject

The TCanvasPoint class represents the X/Y floating-point coordinates of a canvas point. It is useful when used with the TCanvasElement class for calculating coordinates for drawing and fill operations on a canvas.

Properties	Methods	Events
X	Assign	
Y	Clear	
	Create	
	Equals	
	Offset	

TCanvasPoint.X Property

```
property X: Double
```

Specifies the horizontal position of the point.

TCanvasPoint.Y Property

```
property Y: Double
```

Specifies the vertical position of the point.

TCanvasPoint.Assign Method

```
procedure Assign(APoint: TCanvasPoint)

procedure Assign(AX,AY: Double)
```

Use this method to assign a source point to the point instance.

TCanvasPoint.Clear Method

```
procedure Clear
```

Use this method to set both of the point coordinates to 0.

TCanvasPoint.Create Method

```
constructor Create(AX,AY: Double)
```

Use this method to create a new instance of the TCanvasPoint class using the provided X and Y coordinates.

TCanvasPoint.Equals Method

```
function Equals(APoint: TCanvasPoint): Boolean
```

Use this method to test if two points have the same coordinates.

TCanvasPoint.Offset Method

```
procedure Offset(AX,AY: Double)
```

Use this method to offset the point. Offsetting a point increments or decrements its X and Y properties using the AX and AY parameters, respectively.

10.38 TCanvasRect Component

Unit: WebUI

Inherits From TObject

The TCanvasRect class represents a canvas rectangle using floating-point coordinates. It is useful when used with the TCanvasElement class for calculating rectangle coordinates for drawing and fill operations on a canvas.

Properties	Methods	Events
Bottom	Anchor	
Empty	Assign	
Height	Clear	
Left	Create	
Right	Equals	
Top	Fit	
Width	Inflate	
	Interpolate	
	Normalize	
	Offset	
	Scale	
	Union	

TCanvasRect.Bottom Property

```
property Bottom: Double
```

Specifies the bottom Y coordinate of the rectangle.

TCanvasRect.Empty Property

```
property Empty: Boolean
```

Indicates whether the rectangle is empty. A rectangle is considered empty if its width or height is 0.

TCanvasRect.Height Property

```
property Height: Double
```

Indicates the height of the rectangle (read-only).

TCanvasRect.Left Property

```
property Left: Double
```

Specifies the left X coordinate of the rectangle.

TCanvasRect.Right Property

```
property Right: Double
```

Specifies the right X coordinate of the rectangle.

TCanvasRect.Top Property

```
property Top: Double
```

Specifies the top Y coordinate of the rectangle.

TCanvasRect.Width Property

```
property Width: Double
```

Indicates the width of the rectangle (read-only).

TCanvasRect.Anchor Method

```
procedure Anchor
```

Use this method to anchor the rectangle. Anchoring a rectangle changes its Left and Top properties to 0, and adjusts the Right and Bottom properties accordingly so that the rectangle retains its same Width and Height.

TCanvasRect.Assign Method

```
procedure Assign(ARect: TCanvasRect)

procedure Assign(ALeft,ATop,ARight,ABottom: Double)
```

Use this method to assign a source rectangle, or source rectangle coordinates, to the rectangle instance.

TCanvasRect.Clear Method

```
procedure Clear
```

Use this method to set all of the rectangle coordinates to 0.

TCanvasRect.Create Method

```
constructor Create(ALeft,ATop,ARight,ABottom: Double=0)
```

Use this method to create a new instance of the TCanvasRect class. The optional ALeft, ATop, ARight, and ABottom parameters will initialize the instance with the provided coordinates.

TCanvasRect.Equals Method

```
function Equals(ARect: TCanvasRect): Boolean
```

Use this method to test if two rectangles have the same coordinates.

> **Note**
> If the compared coordinates are not whole numbers and contain fractions, then the comparison results may not be entirely as expected due to the possibility of unequal fractional portions for each coordinate.

TCanvasRect.Fit Method

```
procedure Fit(AWidth,AHeight: Double)
```

Use this method to proportionally adjust the width and height of the rectangle so that it fits within the specified AWidth and AHeight parameters.

TCanvasRect.Inflate Method

```
procedure Inflate(ALeft,ATop,ARight,ABottom: Double)
```

Use this method to inflate the rectangle. Inflating a rectangle increments or decrements its Left, Top, Right, and Bottom properties using the specified parameters.

TCanvasRect.Interpolate Method

```
procedure Interpolate(ARect: TCanvasRect; AAmount: Double)
```

Use this method to calculate a new rectangle using the difference between the coordinates of the rectangle and a source rectangle multiplied by a specified distance amount. This calculation is useful for moving a rectangle along a given linear path between a source rectangle and a destination rectangle.

TCanvasRect.Normalize Method

```
procedure Normalize
```

Use this method to normalize the rectangle. Normalizing a rectangle changes (if necessary) its Left, Top, Right, and Bottom properties so that the Left property is less than or equal to the Right property and the Top property is less than or equal to the Bottom property.

TCanvasRect.Offset Method

```
procedure Offset(ALeft: Double; ATop: Double)
```

Use this method to offset the rectangle. Offsetting a rectangle increments or decrements its Left and Top properties using the ALeft and ATop parameters, respectively.

TCanvasRect.Scale Method

```
procedure Scale(ARatio: Double)
```

Use this method to proportionally scale the coordinates of the rectangle using the specified ARatio parameter.

TCanvasRect.Union Method

```
procedure Union(ARect: TCanvasRect)
```

Use this method to union the coordinates of the rectangle with another rectangle. A union operation is a max operation on each pair of coordinates, taking the smallest of the two rectangles' Left and Top properties, and the largest of the two rectangles' Right and Bottom properties.

10.39 TCaptionBarControl Component

Unit: WebCtnrs

Inherits From TControl

The TCaptionBarControl control is the base class for caption bar controls, and contains all of the core panel functionality in the form of public methods and protected properties/events that descendant classes can use to create customized caption bar controls.

Properties	Methods	Events

10.40 TCheckBox Component

Unit: WebBtns

Inherits From TStateButtonControl

The TCheckBox component represents a checkbox control. A checkbox control is a state control that allows the user to toggle a selection state on and off by using a mouse click, or by pushing the spacebar or enter key.

Properties	Methods	Events
AutoWidth		OnAnimationComplete
Caption		OnAnimationsComplete
Cursor		OnChange
DataColumn		OnClick
DataSet		OnEnter
Enabled		OnExit
Font		OnHide
Hint		OnKeyDown
ReadOnly		OnKeyPress
SelectionState		OnKeyUp
TabOrder		OnMouseDown
TabStop		OnMouseEnter
ValueSelected		OnMouseLeave
ValueUnselected		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TCheckBox.AutoWidth Property

```
property AutoWidth: Boolean
```

Specifies whether the width of the check box should be automatically set based upon the Caption and Font properties.

TCheckBox.Caption Property

```
property Caption: String
```

Specifies the caption for the control.

Note
The caption area of a control is also clickable with the mouse or touch interface.

TCheckBox.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TCheckBox.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TCheckBox.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TCheckBox.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TCheckBox.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TCheckBox.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TCheckBox.ReadOnly Property

```
property ReadOnly: Boolean
```

Specifies whether the control's input value can be modified by the user. The default value is False.

> **Note**
> The input value can always be programmatically modified.

TCheckBox.SelectionState Property

```
property SelectionState: TSelectionState
```

Specifies the selection state of the control.

Note
The ssIndeterminate selection state can only be set programmatically. When the user toggles the selection for the control, it will alternate between the ssSelected and ssUnselected selection states.

TCheckBox.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TCheckBox.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TCheckBox.ValueSelected Property

```
property ValueSelected: String
```

Specifies the textual value to use for the selected state when reading and writing data to and from the data column that the control is bound to. The default value is 'True'.

TCheckBox.ValueUnselected Property

```
property ValueUnselected: String
```

Specifies the textual value to use for the unselected state when reading and writing data to and from the data column that the control is bound to. The default value is 'False'.

TCheckBox.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TCheckBox.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TCheckBox.OnChange Event

```
property OnChange: TNotifyEvent
```

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TCheckBox.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TCheckBox.OnEnter Event

```
property OnEnter: TNotifyEvent
```

This event is triggered when the control obtains input focus.

TCheckBox.OnExit Event

```
property OnExit: TNotifyEvent
```

This event is triggered when the control loses input focus.

TCheckBox.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TCheckBox.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when the control has input focus and the user presses a key or key combination.

TCheckBox.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TCheckBox.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when the control has input focus and the user releases a key or key combination.

TCheckBox.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TCheckBox.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TCheckBox.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TCheckBox.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TCheckBox.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TCheckBox.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TCheckBox.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TCheckBox.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TCheckBox.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TCheckBox.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TCheckBox.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TCheckBox.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.41 TCollection Component

Unit: WebCore

Inherits From TPersistent

The TCollection class represents a list of TCollectionItem descendant class instances, and is used to create lists of instances that can be automatically persisted at design-time and run-time. For example, the Elevate Web Builder component library uses the TCollection class as the ancestor for the TDataColumns class so that TDataSet instance columns can be persisted with forms.

Note
Collections always own all items in the collection and will automatically dispose of all items in the collection when the collection is destroyed.

Properties	Methods	Events
Count	Add	OnChanged
ItemClass	BeginUpdate	
Items	Clear	
	Create	
	Delete	
	EndUpdate	
	FindItemByID	
	GetNames	
	IndexOf	
	Insert	
	Move	

TCollection.Count Property

```
property Count: Integer
```

Indicates the number of items in the collection.

TCollection.ItemClass Property

```
property ItemClass: TCollectionItemClass
```

Indicates the TCollectionItemClass class type that the collection will use when creating new items for the collection.

TCollection.Items Property

```
property Items[AIndex: Integer]: TCollectionItem

property Items[const AName: String]: TCollectionItem
```

Accesses items in the collection by their Index property or by their Name.

TCollection.Add Method

```
function Add: TCollectionItem
```

Use this method to add a new item to the collection. The return value is the new collection item instance, and its class type is determined by the ItemClass property.

TCollection.BeginUpdate Method

```
procedure BeginUpdate
```

Use this method to begin a batch update to the collection. Batch updates are useful in situations where many changes need to be made to the collection, and triggering the OnChanged event on every change would result in performance issues. This method is reference-counted and every time it is called, an internal counter is incremented. Every time the EndUpdate method is called, the counter is decremented. Once the counter reaches zero, the OnChanged event will be triggered.

TCollection.Clear Method

```
procedure Clear
```

Use this method to remove all items from the collection, freeing the item instances in the process.

TCollection.Create Method

```
constructor Create(AItemClass: TCollectionItemClass)
```

Use this method to create a new instance of the TCollection class. The AItemClass parameter indicates the class type that the collection should use when creating new item instances.

TCollection.Delete Method

```
procedure Delete(AIndex: Integer)
```

Use this method to delete an item from the collection by its Index property, freeing the item instance in the process.

TCollection.EndUpdate Method

```
procedure EndUpdate
```

Use this method to end a batch update to the collection. Batch updates are useful in situations where many changes need to be made to the collection, and triggering the OnChanged event on every change would result in performance issues. This method is reference-counted and every time it is called, an internal counter is decremented. Every time the BeginUpdate method is called, the counter is incremented. Once the counter reaches zero, the OnChanged event will be triggered.

TCollection.FindItemByID Method

```
function FindItemByID(AID: Integer): TCollectionItem
```

Use this method to search for an item in the collection by its ID property. An item's ID is an integer identifier that uniquely identifies each item instance.

TCollection.GetNames Method

```
function GetNames: array of String
```

Use this method to get a list of the names of the items in the collection.

TCollection.IndexOf Method

```
function IndexOf(AItem: TCollectionItem): Integer

function IndexOf(const AName: String): Integer
```

Use this method to search for an item in the collection by its Index property.

TCollection.Insert Method

```
function Insert(AIndex: Integer): TCollectionItem
```

Use this method to insert a new item into the collection at a specific index. The return value is the new collection item instance, and its class type is determined by the ItemClass property.

TCollection.Move Method

```
procedure Move(Source: Integer; Dest: Integer)
```

Use this method to move an item from one index position to another index position. The item's Index will be updated accordingly.

TCollection.OnChanged Event

```
property OnChanged: TNotifyEvent
```

This event is triggered when a collection item is added or deleted, or if any collection item's name is changed.

10.42 TCollectionItem Component

Unit: WebCore

Inherits From TPersistent

The TCollectionItem class represents an item in a TCollection class instance.

Properties	Methods	Events
Collection	Create	
ID		
Index		
Name		
Tag		

TCollectionItem.Collection Property

```
property Collection: TCollection
```

Indicates the collection instance that the item belongs to.

TCollectionItem.ID Property

```
property ID: Integer
```

Indicates the unique ID of the item. The ID is automatically assigned by the Collection when the item is created.

TCollectionItem.Index Property

```
property Index: Integer
```

Indicates the index of the item in the Collection. You can modify this property in order to change the index of the item in the collection.

TCollectionItem.Name Property

```
property Name: TCollectionItemName
```

Specifies the name of the item. A name is not required to be unique, by default, but TCollection descendant classes may enforce uniqueness of item names.

TCollectionItem.Tag Property

```
property Tag: Integer
```

TCollectionItem.Create Method

```
constructor Create(ACollection: TCollection)
```

Use this method to create a new instance of the TCollectionItem class. The ACollection parameter indicates the collection instance that will contain the item instance.

10.43 TComponent Component

Unit: WebCore

Inherits From TPersistent

The TComponent class represents a component and is the base class of all components in Elevate Web Builder (visual and non-visual). This class includes functionality for ownership of other components and event registration/notification, which simplifies component destruction by allowing the TComponent class to automatically destroy all owned components.

Properties	Methods	Events
Component	Create	
ComponentCount	FindComponent	
Data	Register	
Destroying	UnRegister	
Name		
Owner		
Tag		

TComponent.Component Property

```
property Component[Index: Integer]: TComponent
```

Allows indexed access to all components owned by the component. If a component is created with a specific owner, then the component instance that is created is automatically added to the owner component's Component property. When an owner component is destroyed, all owned components are also automatically destroyed at the same time.

TComponent.ComponentCount Property

```
property ComponentCount: Integer
```

Indicates the number of components owned by the component.

TComponent.Data Property

```
property Data: TObject
```

Can be used to associate any TObject or descendant class instance with the component.

TComponent.Destroying Property

```
property Destroying: Boolean
```

Indicates that the component is in the process of being destroyed, meaning that its Destroy destructor has begun executing.

TComponent.Name Property

```
property Name: TComponentName
```

Specifies the name of the component. Components that are placed on a form must have a name that is unique within the context of the containing form.

TComponent.Owner Property

```
property Owner: TComponent
```

Indicates the owner of the component, or nil if the component does not have an owner.

TComponent.Tag Property

```
property Tag: Integer
```

Can be used to associate any integer value with the component.

TComponent.Create Method

```
constructor Create(AOwner: TComponent)
```

Use this method to create a new instance of the TComponent class. The AOwner parameter indicates the component that will own the instance, or nil if the component has no owner. The owner of the component will automatically free the component instance when the owner is freed.

TComponent.FindComponent Method

```
function FindComponent(const Value: String; Recurse:
      Boolean=False): TComponent
```

Use this method to find an owned component by its name. The optional Recurse parameter indicates whether the search should be recursive and include components owned by the owned components, etc.

TComponent.Register Method

```
procedure Register(Component: TComponent)
```

Registers a component with another component so that the calling component receives component notification messages.

For example, a component can register itself with another component so that it can be notified when the component is destroyed. This is useful in situations where a component instance needs to know when another component instance is destroyed so that it can remove any references to the component instance being destroyed.

TComponent.UnRegister Method

```
procedure UnRegister(Component: TComponent)
```

Unregisters a component from another component so that the calling component no longer receives any notification messages.

10.44 TConstraint Component

Unit: WebUI

Inherits From TElementAttribute

The TConstraint class represents a set of width and height constraints for a UI element or control.

Properties	Methods	Events
Height	SetToDefault	
Width		

TConstraint.Height Property

```
property Height: Integer
```

Specifies the height constraint. A value of zero (the default) indicates that no height constraint is in effect.

TConstraint.Width Property

```
property Width: Integer
```

Specifies the width constraint. A value of zero (the default) indicates that no width constraint is in effect.

TConstraint.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the constraint's properties to their default values.

10.45 TConstraints Component

Unit: WebUI

Inherits From TElementAttribute

The TConstraints class represents the minimum/maximum width and height constraints for a UI element or control. Width and height constraints ensure that any attempts to resize the UI element or control is subject to the defined constraints.

Properties	Methods	Events
Max	SetToDefault	
Min		

TConstraints.Max Property

```
property Max: TConstraint
```

Specifies the maximum width and height constraints.

TConstraints.Min Property

```
property Min: TConstraint
```

Specifies the minimum width and height constraints.

TConstraints.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the constraints' properties to their default values.

10.46 TContentLayout Component

Unit: WebCtrls

Inherits From TPersistent

The TContentLayout class represents the layout of content in UI elements, and is used to apply layouts similar to that of background images to a content element in controls like the TImage control.

Properties	Methods	Events
Height		
Left		
Position		
Size		
Top		
Width		

TContentLayout.Height Property

```
property Height: Integer
```

Specifies the height of the element. If the actual height of the element is different than this value, then the element height is stretched/contracted accordingly.

Note
This property is only valid when the Size is csSpecified.

TContentLayout.Left Property

```
property Left: Integer
```

Specifies the left position of the element.

Note
This property is only valid when the Position is cpSpecified.

TContentLayout.Position Property

```
property Position: TContentPosition
```

Specifies how the element should be positioned within the UI element or control.

TContentLayout.Size Property

```
property Size: TContentSize
```

Specifies how the element should be sized within the UI element or control.

TContentLayout.Top Property

```
property Top: Integer
```

Specifies the top position of the element.

Note
This property is only valid when the Position is cpSpecified.

TContentLayout.Width Property

```
property Width: Integer
```

Specifies the width of the element. If the actual width of the element is different than this value, then the element width is stretched/contracted accordingly.

Note
This property is only valid when the Size is csSpecified.

10.47 TControl Component

Unit: WebCtrls

Inherits From TInterfaceController

The TControl component is the base class for all visual controls and provides layout and dimensional functionality, as well as functionality for handling child controls, mouse, keyboard, and touch events, focus, and visibility.

Properties	Methods	Events
ActiveControl	BeginUpdate	
AlwaysOnTop	BringToFront	
Animations	CanFocus	
ClientHeight	DefineLayout	
ClientID	EndUpdate	
ClientWidth	Float	
Constraints	ForceUpdate	
ControlCount	GetControlNames	
Controls	Hide	
DisplayOrder	IndexOfControl	
Focused	MakeVisible	
Height	Minimize	
InUpdate	RemoveFocus	
Layout	Restore	
LayoutOrder	ScrollBy	
Left	SendToBack	
Margins	SetFocus	
Minimized	Show	
Parent	SlideTo	
ScrollHeight	Tab	
ScrollLeft		
ScrollTop		
ScrollWidth		
SurfaceLeft		
SurfaceTop		
Top		
Visible		
VisibleControlCount		
VisibleControls		
Width		

TControl.ActiveControl Property

```
property ActiveControl: TControl
```

Indicates which child control, if any, has focus in the current control.

TControl.AlwaysOnTop Property

```
property AlwaysOnTop: Boolean
```

Specifies that the control should always be visually placed on top of any other control within the same container control.

TControl.Animations Property

```
property Animations: TAnimations
```

Specifies the animations for the control.

TControl.ClientHeight Property

```
property ClientHeight: Integer
```

Indicates the height of the client rectangle for the control.

TControl.ClientID Property

```
property ClientID: String
```

Specifies the unique DOM ID to assign to the control's client element. This is useful for situations where you need to identify the element from external Javascript code. By default, Elevate Web Builder never assigns DOM IDs to elements because it doesn't need or use them to identify UI elements.

TControl.ClientWidth Property

```
property ClientWidth: Integer
```

Indicates the width of the client rectangle for the control.

TControl.Constraints Property

```
property Constraints: TConstraints
```

Indicates the dimensional constraints for the control.

TControl.ControlCount Property

```
property ControlCount: Integer
```

Indicates the total number of child controls for the control.

TControl.Controls Property

```
property Controls[AIndex: Integer]: TControl
```

Accesses a child control by its index position in the child controls, which are ordered by their LayoutOrder property.

TControl.DisplayOrder Property

```
property DisplayOrder: Integer
```

Specifies the display order, or visual stacking order, of the control within its parent container control.

TControl.Focused Property

```
property Focused: Boolean
```

Indicates whether the control currently has focus.

TControl.Height Property

```
property Height: Integer
```

Indicates the current height of the control, and can be used to specify the **defined** height for the control. The defined height is the last value that was directly assigned to the Height property. Layout property changes may affect the value returned by the Height property.

TControl.InUpdate Property

```
property InUpdate: Boolean
```

Indicates whether the control, or any of its parent controls, is currently in a batch update. A control is in a batch update if the BeginUpdate method is called on the control or any of its parent controls. When a control is in a batch update, it doesn't apply any changes to any of its properties until the EndUpdate method is called on the control or any of its parent controls, and the InUpdate property returns False.

Note
Updates are reference-counted so calls to the BeginUpdate method increment the reference count, and calls to the EndUpdate method decrement the reference count.

TControl.Layout Property

```
property Layout: TLayout
```

Specifies the layout for the control.

TControl.LayoutOrder Property

```
property LayoutOrder: Integer
```

Specifies the layout order of the control. The layout order determines the order in which the child controls of a container control are positioned and sized using the layout management functionality for controls.

TControl.Left Property

```
property Left: Integer
```

Indicates the current left position of the control, and can be used to specify the **defined** left position for the control. The defined left position is the last value that was directly assigned to the Left property. Layout property changes may affect the value returned by the Left property.

TControl.Margins Property

```
property Margins: TMargins
```

Specifies the margins to be used for the control when the control is being positioned/sized using the layout management functionality.

TControl.Minimized Property

```
property Minimized: Boolean
```

Indicates whether the control is minimized. A control can be minimized using the Minimize method, and restored using the Restore method.

TControl.Parent Property

```
property Parent: TControl
```

Specifies the parent control that contains the control, or nil if the control has no parent.

TControl.ScrollHeight Property

```
property ScrollHeight: Integer
```

Indicates the total height of the control's content and/or its child controls. If a control's content height is greater than its Height property, then you can use the ScrollTop property to programmatically scroll the control vertically, or to find out the current vertical scroll position.

TControl.ScrollLeft Property

```
property ScrollLeft: Integer
```

Specifies the horizontal scroll position for the control. If a control's ScrollWidth property is greater than its Width property, then you can use this property to programmatically scroll the control horizontally, or to find out the current horizontal scroll position.

TControl.ScrollTop Property

```
property ScrollTop: Integer
```

Specifies the vertical scroll position for the control. If a control's ScrollHeight property is greater than its Height property, then you can use this property to programmatically scroll the control vertically, or to find out the current vertical scroll position.

TControl.ScrollWidth Property

```
property ScrollWidth: Integer
```

Indicates the total width of the control's content and/or its child controls. If a control's content width is greater than its Width property, then you can use the ScrollLeft property to programmatically scroll the control horizontally, or to find out the current horizontal scroll position.

TControl.SurfaceLeft Property

```
property SurfaceLeft: Integer
```

Indicates the current left position of the control, relative to the application surface.

The application surface is represented by the Surface property of the global Application variable in the WebForms unit.

TControl.SurfaceTop Property

```
property SurfaceTop: Integer
```

Indicates the current top position of the control, relative to the application surface.

The application surface is represented by the Surface property of the global Application variable in the WebForms unit.

TControl.Top Property

```
property Top: Integer
```

Indicates the current top position of the control, and can be used to specify the **defined** top position for the control. The defined top position is the last value that was directly assigned to the Top property. Layout property changes may affect the value returned by the Top property.

TControl.Visible Property

```
property Visible: Boolean
```

Indicates whether the control is visible or not. Setting this property to False causes the Hide method to be called, whereas setting this property to True causes the Show method to be called. The default value is True.

TControl.VisibleControlCount Property

```
property VisibleControlCount: Integer
```

Indicates the total number of visible child controls for the control. A visible child control is one whose Visible property is True.

TControl.VisibleControls Property

```
property VisibleControls[AIndex: Integer]: TControl
```

Accesses a visible child control by its index position in the child controls, which are ordered by their LayoutOrder property.

TControl.Width Property

```
property Width: Integer
```

Indicates the current width of the control, and can be used to specify the **defined** width for the control. The defined width is the last value that was directly assigned to the Width property. Layout property changes may affect the value returned by the Width property.

TControl.BeginUpdate Method

```
procedure BeginUpdate
```

Use this method to begin a batch update on a control. A control is in a batch update if the BeginUpdate method is called on the control, or any of its parent controls. When a control is in a batch update, it doesn't apply any changes to any of its properties until the EndUpdate method is called on the control or any of its parent controls, and the InUpdate property returns False.

Note
Updates are reference-counted so calls to the BeginUpdate method increment the reference count, and calls to the EndUpdate method decrement the reference count.

TControl.BringToFront Method

```
procedure BringToFront
```

Use this method to bring the control to the front of the visual stacking order. A control in the front will have a DisplayOrder property equal to one less than the number of child controls in the parent container control.

TControl.CanFocus Method

```
function CanFocus: Boolean
```

Use this method to determine if the current control can acquire input focus. Invisible and disabled controls always return False when this method is called.

TControl.DefineLayout Method

```
procedure DefineLayout
```

Use this method to assign the current bounds of the control to the Left, Top, Width, and Height properties of the control.

This method is useful in situations where a control has been positioned or sized according to its Layout properties, but you wish to have the bounds persist even after modifying the layout properties so that they no longer position or size the control in the same way.

TControl.EndUpdate Method

```
procedure EndUpdate
```

Use this method to end a batch update on a control. A control is in a batch update if the BeginUpdate method is called on the control or any of its parent controls. When a control is in a batch update, it doesn't apply any changes to any of its properties until the EndUpdate method is called on the control or any of its parent controls, and the InUpdate property returns False.

> **Note**
> Updates are reference-counted so calls to the BeginUpdate method increment the reference count, and calls to the EndUpdate method decrement the reference count.

TControl.Float Method

```
procedure Float
```

Use this method to parent the control to the application surface and reset any layout properties set for the control. This method is useful as the first step when dragging controls around the application surface.

The application surface is represented by the Surface property of the global Application variable in the WebForms unit.

TControl.ForceUpdate Method

```
procedure ForceUpdate
```

Use this method to force the control to perform a layout update when in the middle of a BeginUpdate/EndUpdate block. Normally, a control will not update its layout bounds when an update block is in effect. A forced layout update will ensure that the control's layout bounds are updated to reflect any layout changes applied to the control within the update block.

TControl.GetControlNames Method

```
function GetControlNames(AControlClass: TControlClass): array of
      String
```

Use method to get a list of the names of the child controls for the control.

TControl.Hide Method

```
procedure Hide
```

Use this method to hide the control. After calling this method, the Visible property will be set to False.

TControl.IndexOfControl Method

```
function IndexOfControl(AControl: TControl; AControlClass:
      TControlClass=nil): Integer
```

Use this method to determine the index of a child control within the Controls property. You can, optionally, also pass a TControlClass type to limit the search to a particular TControl class type.

TControl.MakeVisible Method

```
procedure MakeVisible(X,Y: Boolean=True)
```

Use this method to ensure that the control is visible within the client rectangle of its parent control.

TControl.Minimize Method

```
procedure Minimize
```

Use this method to minimize the control. The minimization of a control is a control-specific operation, so the results of calling this method will depend upon the control class of the control being minimized.

TControl.RemoveFocus Method

```
procedure RemoveFocus
```

Use this method to remove focus from the control. If the control is not focused, then this method does nothing.

TControl.Restore Method

```
procedure Restore
```

Use this method to restore a minimized control back to its original dimensions.

TControl.ScrollBy Method

```
procedure ScrollBy(X,Y: Integer)
```

If a control's content width and/or height is greater than its Width and Height properties, then you can use this method to scroll the contents of the control horizontally, vertically, or both. The X and Y values represent the number of pixels by which to scroll the contents, and may be negative values for scrolling backward.

TControl.SendToBack Method

```
procedure SendToBack
```

Use this method to send the control to the back of the visual stacking order. A control in the back will have a DisplayOrder property equal to 0.

TControl.SetFocus Method

```
procedure SetFocus
```

Use this method to set focus to the control. Use the Focused property to determine if the control was actually able to obtain the focus.

The ability of a control to obtain the input focus is dependent upon:

- Whether the control is focusable
- Whether the control is visible
- Whether its parent container control is visible

TControl.Show Method

```
procedure Show
```

Use this method to show the control. After calling this method, the Visible property will be set to True.

TControl.SlideTo Method

```
procedure SlideTo(X,Y: Integer; AnimationStyle: TAnimationStyle;
      AnimationDuration: Integer; Fade: Boolean=False)
```

Use this method to slide the control to a specific location using a specific animation style/duration and, optionally, with a corresponding fade (hides the control).

Note
This method updates the properties of the Left, Top, and, optionally, the Visible animation primitives for the control's Animations property.

TControl.Tab Method

```
procedure Tab(Backward: Boolean=False)
```

Use this method to navigate to the next or prior control after/before the currently-focused control, based upon the tabbing order for the container control.

The Backward parameter determines if the navigation occurs forwards or backwards in the tabbing order.

10.48 TCookies Component

Unit: WebComps

Inherits From TObject

The TCookies class encapsulates the cookie functionality in the web browser.

Note
The component library includes a global instance variable of this class called Cookies in the WebComps unit that should be used instead of creating new instances of the class.

Warning
Do not attempt to store more than 4k of data in a single cookie (name and value combined). If you need to store more data than that, then you should use the Local Storage functionality instead.

Properties	Methods	Events
Count	Clear	
Enabled	Create	
Items	Exists	
	Refresh	
	Set	

TCookies.Count Property

```
property Count: Integer
```

Indicates the number of cookies defined.

TCookies.Enabled Property

```
property Enabled: Boolean
```

Indicates whether cookies are enabled or not in the web browser. If this property is False, then the cookie functionality is disabled and the TCookies object will not function properly.

TCookies.Items Property

```
property Items[Index: Integer]: String

property Items[const AName: String]: String
```

Accesses all defined cookies by index or by name.

TCookies.Clear Method

```
procedure Clear(const AName: String; const Path: String='';
      const Domain: String=''; Secure: Boolean=False)
```

Use this method to clear an existing cookie. The Name parameter is the cookie name, the Path parameter is the relative path under the domain that the cookie applies to, the Domain parameter is the web site domain that the cookie applies to, and the Secure parameter indicates whether the cookie applies to a secure domain/path (https://) or a normal domain/path (http://).

Note

Use the Exists method to determine if a cookie exists before trying to clear the cookie.

TCookies.Create Method

```
constructor Create
```

Use this method to create a new instance of the TCookies class.

TCookies.Exists Method

```
function Exists(const AName: String): Boolean
```

Use this method to determine if the specified cookie exists.

TCookies.Refresh Method

```
procedure Refresh
```

Use this method to reload the cookies from the web browser. This is useful for situations where an embedded HTML document in a TBrowser control has modified the cookies and these modifications need to be reflected in the global TCookies instance.

> **Note**
> Cookies are cached in a TCookies instance and do not immediately reflect changes to the web browser cookies done via third party JavaScript code or embedded documents. This is why this method is necessary.

TCookies.Set Method

```
procedure Set(const AName: String; const Value: String; MaxAge:
      Integer=-1; const Path: String=''; const Domain: String='';
      Secure: Boolean=False)
```

Use this method to set a cookie value. The Name parameter is the cookie name, the MaxAge parameter specifies the length of time, in seconds, that the browser should cache the cookie, the Path parameter is the relative path under the domain that the cookie applies to, the Domain parameter is the web site domain that the cookie applies to, and the Secure parameter indicates whether the cookie applies to a secure domain/path (https://) or a normal domain/path (http://).

Note
Using -1 (or any value less than 0) as the MaxAge parameter causes the cookie to be a session-only cookie that is automatically deleted by the web browser when it is closed/shut down.

10.49 TCorner Component

Unit: WebUI

Inherits From TElementAttribute

The TCorners class represents the horizontal and vertical radius of a border corner for a UI element or control.

Properties	Methods	Events
HorzRadius	SetToDefault	
VertRadius		

TCorner.HorzRadius Property

```
property HorzRadius: Integer
```

Specifies the horizontal radius.

TCorner.VertRadius Property

```
property VertRadius: Integer
```

Specifies the vertical radius.

TCorner.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the corner's properties to their default values.

10.50 TCorners Component

Unit: WebUI

Inherits From TElementAttribute

The TCorner class represents the horizontal and vertical radii of the border corners for a UI element or control.

Properties	Methods	Events
BottomLeft	SetToDefault	
BottomRight		
TopLeft		
TopRight		

TCorners.BottomLeft Property

```
property BottomLeft: TCorner
```

Specifies the bottom-left corner radius.

TCorners.BottomRight Property

```
property BottomRight: TCorner
```

Specifies the bottom-right corner radius.

TCorners.TopLeft Property

```
property TopLeft: TCorner
```

Specifies the top-left corner radius.

TCorners.TopRight Property

```
property TopRight: TCorner
```

Specifies the top-right corner radius.

TCorners.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the corners' properties to their default values.

10.51 TDatabase Component

Unit: WebData

Inherits From TComponent

The TDatabase component represents a database container for all datasets in an application and provides properties and methods for loading/saving datasets and using transactions to modify the datasets. An instance of the TDatabase component called **Database** is automatically created by the component library at application startup, so further instances of the TDatabase component should not be created.

Properties	Methods	Events
AuthenticationMethod	CancelPendingRequests	AfterCommit
AutoTransactions	Commit	AfterRollback
BaseURL	GetTransactions	BeforeCommit
DatabaseName	LoadColumns	BeforeRollback
DataSetCount	LoadRows	OnCommitError
DataSets	LoadTransactions	OnCreate
InTransaction	RetryPendingRequests	OnDestroy
NumPendingRequests	Rollback	OnRollbackError
Params	StartTransaction	
Password		
Timeout		
TransactionLevel		
UserName		

TDatabase.AuthenticationMethod Property

```
property AuthenticationMethod: TAuthenticationMethod
```

Specifies the authentication method to use when sending authentication information to the web server along with dataset requests.

TDatabase.AutoTransactions Property

```
property AutoTransactions: Boolean
```

Specifies whether transactions will be automatically started and committed or rolled back as rows are inserted, updated, and deleted in the datasets owned by the database. The default value is True.

Note

Automatic transactions are implicitly nested. If you insert a row in one table, and then edit a row in another table, the TransactionLevel property will be 1.

TDatabase.BaseURL Property

```
property BaseURL: String
```

Specifies the base URL used for all dataset requests. The default value is "datasets".

TDatabase.DatabaseName Property

```
property DatabaseName: String
```

Specifies the name of the database to use with a URL when building a web server request for loading the columns or rows of a dataset from the web server application, or when building a web server request for committing a transaction to the web server application.

TDatabase.DataSetCount Property

```
property DataSetCount: Integer
```

Indicates the number of datasets that have been created in the application.

TDatabase.DataSets Property

```
property DataSets[Index: Integer]: TDataSet

property DataSets[const Name: String]: TDataSet
```

Accesses all datasets in the application by index or by name.

TDatabase.InTransaction Property

```
property InTransaction: Boolean
```

Indicates whether a transaction is active or not.

TDatabase.NumPendingRequests Property

```
property NumPendingRequests: Integer
```

Indicates the number of pending database load/commit web server requests. If any errors were encountered during a dataset load or database transaction commit operation, then the web server request used with the operation will remain as a pending request. You can use the RetryPendingRequests to retry all pending requests, or the CancelPendingRequests to cancel all pending requests.

TDatabase.Params Property

```
property Params: TStrings
```

Specifies any custom database-specific name/value parameters to use with any dataset requests. These parameters are sent along with the built-in parameters used by the database and datasets, and are useful for specifying additional parameters for the web server application to use.

TDatabase.Password Property

```
property Password: String
```

Specifies the password to use for any LoadColumns, LoadRows, or Commit calls. This property is used in conjunction with the UserName property to provide authentication information to the web server application.

TDatabase.Timeout Property

```
property Timeout: Integer
```

Specifies how long any database request should wait, in milliseconds, for a successful connection to the server before returning an error. The default value is 0, which means to wait a browser-defined number of milliseconds.

TDatabase.TransactionLevel Property

```
property TransactionLevel: Integer
```

Indicates the current transaction level. A value of -1 means that no transaction is active, while a value of 0 or higher indicates that a transaction is active up to N levels of nesting.

TDatabase.UserName Property

```
property UserName: String
```

Specifies the user name to use for any LoadColumns, LoadRows, or Commit calls. This property is used in conjunction with the Password property to provide authentication information to the web server application.

TDatabase.CancelPendingRequests Method

```
procedure CancelPendingRequests
```

Use this method to cancel any pending dataset load or transaction commit requests. You can determine if there are any pending requests by examining the NumPendingRequests property.

TDatabase.Commit Method

```
procedure Commit
```

Use this method to commit the active transaction.

TDatabase.GetTransactions Method

```
function GetTransactions: String
```

Use this method to retrieve all active transaction operations, including any nested transactions, for the database as a JSON string that can be stored using a TPersistentStorage instance.

TDatabase.LoadColumns Method

```
procedure LoadColumns(DataSet: TDataSet)
```

Use this method to load the columns for the specified dataset from the web server. This method will replace all existing columns in the specified dataset with the columns defined in the JSON returned from the request.

TDatabase.LoadRows Method

```
procedure LoadRows(DataSet: TDataSet; Append: Boolean=False)
```

Use this method to load the rows for the specified dataset from the web server. Specify True for the Append parameter to have the rows appended to the dataset. The default behavior is to replace all rows in the dataset with the rows returned by the web server during the execution of this method.

TDatabase.LoadTransactions Method

```
procedure LoadTransactions(const Value: String)
```

Use this method to load transactions operations, including nested transactions, for the database from a JSON string.

Warning
All datasets that were present when the transactions operations were retrieved as a JSON string must be present when using this method to re-load the transaction operations. If a dataset that was updated in a transaction is no longer present, then this method will raise an exception.

TDatabase.RetryPendingRequests Method

```
procedure RetryPendingRequests
```

Use this method to retry any pending dataset load or transaction commit requests. You can determine if there are any pending requests by examining the NumPendingRequests property.

TDatabase.Rollback Method

```
procedure Rollback
```

Use this method to roll back the active transaction.

TDatabase.StartTransaction Method

```
procedure StartTransaction
```

Use this method to start a new transaction.

TDatabase.AfterCommit Event

```
property AfterCommit: TNotifyEvent
```

This event is triggered after the Commit method completes.

TDatabase.AfterRollback Event

```
property AfterRollback: TNotifyEvent
```

This event is triggered after the Rollback method completes.

TDatabase.BeforeCommit Event

```
property BeforeCommit: TDatabaseEvent
```

This event is triggered before the Commit method starts. Return False from the event handler to prevent the commit from occurring.

TDatabase.BeforeRollback Event

```
property BeforeRollback: TDatabaseEvent
```

This event is triggered before the Rollback method starts. Return False from the event handler to prevent the rollback from occurring.

TDatabase.OnCommitError Event

```
property OnCommitError: TDatabaseErrorEvent
```

This event is triggered whenever an error occurs during the execution of the Commit method.

Note
If an event handler is not assigned to this event, then the error will be raised as an exception.

TDatabase.OnCreate Event

```
property OnCreate: TNotifyEvent
```

This event is triggered after the database is created and initialized.

Note
Any design-time components placed on the database will already be instantiated and initialized before this event is triggered.

TDatabase.OnDestroy Event

```
property OnDestroy: TNotifyEvent
```

This event is triggered before the database is destroyed. Use this event to dispose of any instances or resources that may have been allocated in the OnCreate event handler.

TDatabase.OnRollbackError Event

```
property OnRollbackError: TDatabaseErrorEvent
```

This event is triggered whenever an error occurs during the execution of the Rollback method.

Note
If an event handler is not assigned to this event, then the error will be raised as an exception.

10.52 TDataColumn Component

Unit: WebData

Inherits From TCollectionItem

The TDataColumn class represents a column in a TDataSet component and contains functionality for getting and setting column values, tracking modifications, and sorting.

Properties	Methods	Events
AsBoolean	Clear	OnGetText
AsDate		OnSetText
AsDateTime		
AsFloat		
AsInteger		
AsString		
AsTime		
Calculated		
DataType		
Length		
Modified		
Null		
OldValue		
ReadOnly		
Scale		
SortDirection		
SortIndex		
Text		

TDataColumn.AsBoolean Property

```
property AsBoolean: Boolean
```

Gets or sets the column value in the active row as a boolean value. If the column value cannot be expressed as a boolean value, then an exception will be raised.

TDataColumn.AsDate Property

```
property AsDate: DateTime
```

Gets or sets the column value in the active row as a date value. If the column value cannot be expressed as a date value, then an exception will be raised.

TDataColumn.AsDateTime Property

```
property AsDateTime: DateTime
```

Gets or sets the column value in the active row as a date/time value. If the column value cannot be expressed as a date/time value, then an exception will be raised.

TDataColumn.AsFloat Property

```
property AsFloat: Double
```

Gets or sets the column value in the active row as a floating-point value. If the column value cannot be expressed as a floating-point value, then an exception will be raised.

TDataColumn.AsInteger Property

```
property AsInteger: Integer
```

Gets or sets the column value in the active row as an integer value. If the column value cannot be expressed as an integer value, then an exception will be raised.

TDataColumn.AsString Property

```
property AsString: String
```

Gets or sets the column value in the active row as a string value. If the column value cannot be expressed as a string value, then an exception will be raised.

TDataColumn.AsTime Property

```
property AsTime: DateTime
```

Gets or sets the column value in the active row as a time value. If the column value cannot be expressed as a time value, then an exception will be raised.

TDataColumn.Calculated Property

```
property Calculated: Boolean
```

Specifies that the column is calculated via the TDataSet OnCalculateRow event handler, if one exists.

> **Note**
> Any attempt to programmatically modify a calculated column outside of an OnCalculateRow event handler will result in an error.

TDataColumn.DataType Property

```
property DataType: TDataType
```

Gets or sets the data type of the column.

Note
Attempting to change the data type of a column for a dataset that has been opened will result in an exception.

TDataColumn.Length Property

```
property Length: Integer
```

Gets or sets the length of the column. The length should only be set for string and BLOB column types and all other column types should have a length of -1.

Note
Attempting to change the length of a column for a dataset that has been opened will result in an exception.

TDataColumn.Modified Property

```
property Modified: Boolean
```

Indicates if the column has been modified in the active row as part of an insert or update operation.

TDataColumn.Null Property

```
property Null: Boolean
```

Indicates if the column in the active row is NULL.

TDataColumn.OldValue Property

```
property OldValue: TDataValue
```

Provides a reference to the prior version of the column value when updating a row in a dataset.

TDataColumn.ReadOnly Property

```
property ReadOnly: Boolean
```

Specifies that the column is read-only.

Note

Any attempt to programmatically modify a read-only column will result in an error.

TDataColumn.Scale Property

```
property Scale: Integer
```

Gets or sets the scale of the column. The scale should only be set for floating-point column types and all other column types should have a scale of -1.

> **Note**
> Attempting to change the scale of a column for a dataset that has been opened will result in an exception.

TDataColumn.SortDirection Property

```
property SortDirection: TSortDirection
```

Specifies whether the column participates in the active sort for the dataset and, if so, which direction the column should be sorted in.

TDataColumn.SortIndex Property

```
property SortIndex: Integer
```

If the SortDirection property is not equal to sdNone, then this property indicates the position of the column in the active sort for the dataset.

TDataColumn.Text Property

```
property Text: String
```

Gets or sets the column display text in the active row as a string value. Reading this property triggers the OnGetText event for the column and assigning a value to this property triggers the OnSetText event for the column.

TDataColumn.Clear Method

```
procedure Clear
```

Use this method to clear the column value in the active row and set the Null property to True. This will also cause the Modified property to be set to True.

TDataColumn.OnGetText Event

```
property OnGetText: TDataColumnTextEvent
```

This event is triggered when the TDataColumn Text property is read, giving the application a chance to format the resultant text as required. Visual controls that are bound to a column use the Text property to display the data for the column.

TDataColumn.OnSetText Event

```
property OnSetText: TDataColumnTextEvent
```

This event is triggered when the TDataColumn Text property is assigned a value, giving the application a chance to format or parse the text as required before it is actually stored in the column. Visual controls that are bound to a column use the Text property to assign the data for the column as it is modified by the end user.

10.53 TDataColumns Component

Unit: WebData

Inherits From TCollection

The TDataColumns class represents the list of columns in a TDataSet component and contains functionality for managing the columns.

Properties	Methods	Events
Column		

TDataColumns.Column Property

```
property Column[Index: Integer]: TDataColumn

property Column[const Name: String]: TDataColumn
```

Accesses all columns in the dataset by index or by name.

10.54 TDataRow Component

Unit: WebData

Inherits From TDataValues

The TDataRow class represents a row in a TDataSet component. You can access the column values for the current row in a dataset via the TDataSet Columns property.

Properties	Methods	Events

10.55 TDataSet Component

Unit: WebData

Inherits From TComponent

The TDataSet component represents a dataset and includes functionality for opening, loading, navigating, searching, sorting, updating, and closing datasets. Every dataset that is created is automatically added to the TDatabase DataSets property.

Properties	Methods	Events
AutoEdit	BeginControlUpdate	AfterCancel
BOF	Cancel	AfterClose
ColumnCount	CheckBrowseMode	AfterDelete
Columns	Close	AfterInsert
DataSetName	Delete	AfterLoad
Editing	DisableControls	AfterLoadColumns
EOF	Empty	AfterOpen
LocalizeDateTimeColumns	EnableControls	AfterSave
Modified	EndControlUpdate	AfterScroll
OwnerDatabase	Find	AfterUpdate
Params	First	BeforeCancel
RowCount	FreeBookmark	BeforeClose
RowID	GetColumns	BeforeDelete
RowNo	GetRows	BeforeInsert
SortCaseInsensitive	GotoBookmark	BeforeLoad
Sorted	InitFind	BeforeLoadColumns
SortLocaleInsensitive	Insert	BeforeOpen
State	Last	BeforeSave
	LoadColumns	BeforeScroll
	LoadRows	BeforeUpdate
	MoveBy	OnCalculateRow
	MoveTo	OnInitRow
	Next	OnLoadColumnsError
	Open	OnLoadError
	Prior	OnRowChanged
	Save	OnSortChanged
	SaveBookmark	OnStateChange
	Sort	
	Update	

TDataSet.AutoEdit Property

```
property AutoEdit: Boolean
```

Specifies whether the dataset can be automatically edited by the user using data-bound visual controls.

TDataSet.BOF Property

```
property BOF: Boolean
```

Indicates whether the row pointer is on the first row in the set of rows in the dataset. This property is only set to True if the First method is called, if the row pointer attempts to navigate beyond the first row using the Prior method, or if the dataset is empty.

TDataSet.ColumnCount Property

```
property ColumnCount: Integer
```

Indicates the number of columns in the dataset.

TDataSet.Columns Property

```
property Columns: TDataColumns
```

Provides access to the columns in the dataset.

TDataSet.DataSetName Property

```
property DataSetName: String
```

Specifies the name of the dataset to use with a URL when building a web server request for loading the columns or rows of the dataset from the web server application.

TDataSet.Editing Property

```
property Editing: Boolean
```

Indicates whether the dataset is currently inserting or updating the active row via the Insert or Update methods, or if the dataset is currently searching for a row via the Find method.

TDataSet.EOF Property

```
property EOF: Boolean
```

Indicates whether the row pointer is on the last row in the set of rows in the dataset. This property is only set to True if the Last method is called, the row pointer attempts to navigate beyond the last row using the Next method, or if the dataset is empty.

TDataSet.LocalizeDateTimeColumns Property

```
property LocalizeDateTimeColumns: Boolean
```

Specifies whether the dataset will localize date/time column values when converting them to/from strings. The default value is False.

TDataSet.Modified Property

```
property Modified: Boolean
```

Indicates whether the active row in the dataset has been modified since the Insert, Update, or Find methods have been called.

TDataSet.OwnerDatabase Property

```
property OwnerDatabase: TDatabase
```

Indicates the TDatabase instance that owns the dataset.

TDataSet.Params Property

```
property Params: TStrings
```

Specifies any custom dataset-specific name/value parameters to use with any dataset requests for this dataset. These parameters are sent along with the built-in parameters used by the database and datasets, and are useful for specifying additional parameters for the web server application to use.

TDataSet.RowCount Property

```
property RowCount: Integer
```

Indicates the number of rows in the dataset.

TDataSet.RowID Property

```
property RowID: Integer
```

Gets or sets the row ID for the active row in the dataset. If this property is set, then the dataset will automatically navigate to the row that contains the specified row ID, if a row exists with a matching row ID. Each row in a dataset contains an ID that uniquely identifies the row in the dataset.

TDataSet.RowNo Property

```
property RowNo: Integer
```

Gets or sets the row number for the active row in the dataset. If this property is set, then the dataset will automatically navigate to the specified row number, if the row number is greater than 0 and less than or equal to the RowCount property.

Note
The row number for a row is a **logical** construct, which means that it can change as the dataset is sorted, or as rows are inserted or deleted in the dataset.

TDataSet.SortCaseInsensitive Property

```
property SortCaseInsensitive: Boolean
```

Specifies whether or not an active sort on the dataset, as indicated by the Sorted property, should be case-sensitive.

Note
Changing this property will trigger a sort on the dataset if the Sorted property is True.

TDataSet.Sorted Property

```
property Sorted: Boolean
```

Indicates whether any of the Columns in the dataset have their SortDirection property set to sdAscending or sdDescending.

Note

This property does **not** indicate whether the dataset has actually been sorted yet via the Sort method. The sorting is designed this way in order to allow multiple columns to be designated as sort columns without triggering an automatic sort operation each time.

TDataSet.SortLocaleInsensitive Property

```
property SortLocaleInsensitive: Boolean
```

Specifies whether or not an active sort on the dataset, as indicated by the Sorted property, should be locale-sensitive.

Note
Changing this property will trigger a sort on the dataset if the Sorted property is True.

TDataSet.State Property

```
property State: TDataSetState
```

Indicates the state of the dataset.

TDataSet.BeginControlUpdate Method

```
procedure BeginControlUpdate
```

Use this method to disable any notifications to data-bound controls and the triggering of dataset events until the EndControlUpdate method is called. Both of these methods are referenced-counted, so nested calls to the BeginControlUpdate method result in the reference count being incremented, while nested calls to the EndControlUpdate method result in the reference count being decremented. Once the reference count reaches 0, notifications to data-bound controls and the triggering of dataset events will resume.

Note
This method is primarily useful to component developers. The difference between this method and the DisableControls method is that this method also disables the triggering of all dataset events.

TDataSet.Cancel Method

```
procedure Cancel
```

Use this method to cancel any active insert, update, or find operation, changing the State to dsBrowse.

TDataSet.CheckBrowseMode Method

```
function CheckBrowseMode: Boolean
```

Checks to see if the dataset State is set to dsBrowse, and if not, performs the following actions:

- If the dataset is in the process of inserting or updating the active row and has modified the row, then the Save method will be called. If the row has not been modified, then the Cancel method will be called instead.

- If the dataset is in the process of finding a row, then the Cancel method will be called.

TDataSet.Close Method

```
procedure Close
```

Use this method to close an open dataset.

Note
Datasets must be closed in order to modify the Columns in the dataset.

TDataSet.Delete Method

```
procedure Delete
```

Use this method to delete the active row in the dataset. Before the deletion occurs, the BeforeDelete event is triggered, allowing the deletion to be aborted, if necessary. After the deletion occurs, the AfterDelete event is triggered.

Note
If a transaction is active, as indicated by the TDatabase InTransaction property, then the Delete method also logs the current delete operation for inclusion in the current transaction. Please see the Transactions topic for more information.

TDataSet.DisableControls Method

```
procedure DisableControls
```

Use this method to disable any notifications to data-bound controls until the EnableControls method is called, which is useful when performing lengthy operations on datasets that also have controls bound to them. Both of these methods are referenced-counted, so nested calls to the DisableControls method result in the reference count being incremented, while nested calls to the EnableControls method result in the reference count being decremented. Once the reference count reaches 0, notifications to data-bound controls will resume and the controls will actively reflect any changes to the active row in the dataset.

TDataSet.Empty Method

```
procedure Empty
```

Use this method to remove all rows from the dataset.

Warning
This method does not log the removal of the rows, even if a transaction is currently active for the global TDatabase instance.

TDataSet.EnableControls Method

```
procedure EnableControls
```

Use this method to enable any notifications to data-bound controls that were disabled using the DisableControls method. Both of these methods are referenced-counted, so nested calls to the DisableControls method result in the reference count being incremented, while nested calls to the EnableControls method result in the reference count being decremented. Once the reference count reaches 0, notifications to data-bound controls will resume and the controls will actively reflect any changes to the active row in the dataset.

TDataSet.EndControlUpdate Method

```
procedure EndControlUpdate
```

Use this method to enable any notifications to data-bound controls and the triggering of dataset events that were disabled using the BeginControlUpdate method. Both of these methods are referenced-counted, so nested calls to the BeginControlUpdate method result in the reference count being incremented, while nested calls to the EndControlUpdate method result in the reference count being decremented. Once the reference count reaches 0, notifications to data-bound controls and the triggering of dataset events will resume.

Note
This method is primarily useful to component developers. The difference between this method and the EnableControls method is that this method also enables the triggering of all dataset events.

TDataSet.Find Method

```
function Find(NearestMatch: Boolean=False; CaseInsensitive:
      Boolean=False; LocaleInsensitive: Boolean=False): Boolean
```

Use this method to complete a find operation and perform the actual search of the dataset for a row that matches the column values assigned to the various columns after the InitFind method was called. This method will return True if a row is found with the specified column values, or False if not. Specify True for the NearestMatch parameter to cause the search to return the closest match to the desired row, if the desired row cannot be found. Specify True for the CaseInsensitive parameter to cause the search to be executed in a case-insensitive manner for any string columns.

Note

The NearestMatch parameter can only be set to True if there is an active sort on the dataset, as specified by the Sorted property, and the column values that were modified after the InitFind method was called match the columns in the active sort. Also, the CaseInsensitive parameter must also match the SortCaseInsensitive property if there is an active sort on the dataset.

TDataSet.First Method

```
procedure First
```

Use this method to move to the first row in the dataset, taking into account any active sort on the dataset.

TDataSet.FreeBookmark Method

```
procedure FreeBookmark
```

Use this method to free the bookmark last saved via the SaveBookmark method. Bookmark operations are stack-based, so every call to the SaveBookmark method must be followed by a call to the GotoBookmark or FreeBookmark method in order to ensure proper operation.

TDataSet.GetColumns Method

```
function GetColumns: String
```

Use this method to retrieve the defined Columns for the dataset as a JSON string that can either be stored using a TPersistentStorage instance, or sent to the back-end web server using a TServerRequest instance.

TDataSet.GetRows Method

```
function GetRows: String
```

Use this method to retrieve the rows in the dataset as a JSON string that can either be stored using a TPersistentStorage instance, or sent to the back-end web server using a TServerRequest instance.

TDataSet.GotoBookmark Method

```
function GotoBookmark: Boolean
```

Use this method to move the row pointer to the bookmark last saved via the SaveBookmark method, and then free the bookmark. Bookmark operations are stack-based, so every call to the SaveBookmark method must be followed by a call to the GotoBookmark or FreeBookmark method in order to ensure proper operation.

TDataSet.InitFind Method

```
procedure InitFind
```

Use this method to begin searching for a row in the dataset. After modifying the columns that you wish to search on, use the Find method to complete the search for the row.

TDataSet.Insert Method

```
procedure Insert(Append: Boolean=False)
```

Use this method to begin inserting a new row in the dataset. If the Append parameter is True, then the new row will be appended to the end of the dataset, and if the Append parameter is False, then the new row will be inserted at the active row position in the dataset. Before the insert occurs, the BeforeInsert event is triggered, allowing the insertion to be aborted, if necessary. The OnInitRow event is then triggered to allow the new row to be initialized without flagging the row as being modified. After the insertion occurs, the AfterInsert event is triggered.

Use the Save method to complete the insertion of the new row.

TDataSet.Last Method

```
procedure Last
```

Use this method to move to the last row in the dataset, taking into account any active sort on the dataset.

TDataSet.LoadColumns Method

```
procedure LoadColumns(const ColumnData: String)
```

Use this method to load the columns for a dataset from a JSON string. Please see the JSON Reference topic for more information on how the JSON string should be formatted.

> **Note**
> The dataset must be closed or an exception will be raised when this method is called.

TDataSet.LoadRows Method

```
procedure LoadRows(const RowData: String; Append: Boolean=False)
```

Use this method to load the rows for a dataset from a JSON string. Please see the JSON Reference topic for more information on how the JSON string should be formatted.

Note
The dataset must be open or an exception will be raised when this method is called.

TDataSet.MoveBy Method

```
function MoveBy(Value: Integer): Integer
```

Use this method to move N number of rows forward or backward relative to the active row in the dataset, taking into account any active sort on the dataset. If the number of rows passed as the parameter to this method is negative, then the row pointer is moved backward in the dataset. If the number of rows is positive, then the row pointer is moved forward in the dataset.

If the row pointer cannot be moved the specified number of rows without proceeding past the first or last rows in the dataset, then the row pointer will be positioned at the first or last row in the dataset, setting the BOF and/or EOF properties as required.

TDataSet.MoveTo Method

```
procedure MoveTo(Value: Integer)
```

Use this method to move to a specific position in the dataset, taking into account any active sort on the dataset. If the position specified is less than 1 or greater than the number of rows in the dataset, then the row pointer will be positioned at the first or last row in the dataset, setting the BOF and/or EOF properties as required.

TDataSet.Next Method

```
procedure Next
```

Use this method to move the row pointer to the next row in the dataset, taking into account any active sort on the dataset. If the row pointer is already pointing to the last row in the dataset, then the EOF property will be set to True.

TDataSet.Open Method

```
procedure Open
```

Use this method to open a closed dataset.

Note
Datasets must be open in order to load rows into the dataset via the LoadRows method.

TDataSet.Prior Method

```
procedure Prior
```

Use this method to move the row pointer to the prior row in the dataset, taking into account any active sort on the dataset. If the row pointer is already pointing to the first row in the dataset, then the BOF property will be set to True.

TDataSet.Save Method

```
procedure Save
```

Use this method to complete an Insert or Update operation. Before the save occurs, the BeforeSave event is triggered, allowing the save to be aborted, if necessary. After the save occurs, the AfterSave event is triggered.

> **Note**
> If a transaction is active, as indicated by the TDatabase InTransaction property, then the Save method also logs the current insert or update operation for inclusion in the current transaction. Please see the Transactions topic for more information.

TDataSet.SaveBookmark Method

```
procedure SaveBookmark
```

Use this method to save the row pointer as a bookmark. Bookmark operations are stack-based, so every call to the SaveBookmark method must be followed by a call to the GotoBookmark or FreeBookmark method in order to ensure proper operation.

TDataSet.Sort Method

```
procedure Sort
```

Use this method to sort the dataset when the Sorted property is True and sort columns have been specified for the dataset. The TDataColumn SortDirection property setting controls which columns are sorted, and how.

If no columns have a SortDirection property other than sdNone, then this method does nothing.

The SortCaseInsensitive and SortLocaleInsensitive properties control how the sort is performed.

Note

The Sort method only needs to be called once after the sort directions are initially assigned for the columns. After that point, any row operations will automatically maintain the active sort.

TDataSet.Update Method

```
procedure Update
```

Use this method to begin updating the active row in the dataset. Before the update occurs, the BeforeUpdate event is triggered, allowing the update to be aborted, if necessary. After the update occurs, the AfterUpdate event is triggered.

Use the Save method to complete the update of the active row.

TDataSet.AfterCancel Event

```
property AfterCancel: TNotifyEvent
```

This event is triggered after the Cancel method completes.

TDataSet.AfterClose Event

```
property AfterClose: TNotifyEvent
```

This event is triggered after the Close method completes.

TDataSet.AfterDelete Event

```
property AfterDelete: TNotifyEvent
```

This event is triggered after the Delete method completes.

TDataSet.AfterInsert Event

```
property AfterInsert: TNotifyEvent
```

This event is triggered after the Insert method completes.

TDataSet.AfterLoad Event

```
property AfterLoad: TNotifyEvent
```

This event is triggered after the LoadRows method complete.

TDataSet.AfterLoadColumns Event

```
property AfterLoadColumns: TNotifyEvent
```

This event is triggered after the LoadColumns method completes.

TDataSet.AfterOpen Event

```
property AfterOpen: TNotifyEvent
```

This event is triggered after the Open method completes.

TDataSet.AfterSave Event

```
property AfterSave: TNotifyEvent
```

This event is triggered after the Save method completes.

TDataSet.AfterScroll Event

```
property AfterScroll: TNotifyEvent
```

This event is triggered after any navigation operation completes for a dataset. The following TDataSet methods cause the AfterScroll event to be triggered:

First
Prior
Next
Last
MoveBy
MoveTo
Find
Sort
GotoBookmark

TDataSet.AfterUpdate Event

```
property AfterUpdate: TNotifyEvent
```

This event is triggered after the Update method completes.

TDataSet.BeforeCancel Event

```
property BeforeCancel: TDataSetEvent
```

This event is triggered before the Cancel method starts. Return False from the event handler to prevent the cancel from occurring.

TDataSet.BeforeClose Event

```
property BeforeClose: TDataSetEvent
```

This event is triggered before the Close method starts. Return False from the event handler to prevent the close from occurring.

TDataSet.BeforeDelete Event

```
property BeforeDelete: TDataSetEvent
```

This event is triggered before the Delete method starts. Return False from the event handler to prevent the deletion from occurring.

TDataSet.BeforeInsert Event

```
property BeforeInsert: TDataSetEvent
```

This event is triggered before the Insert method starts. Return False from the event handler to prevent the insertion from occurring.

TDataSet.BeforeLoad Event

```
property BeforeLoad: TDataSetEvent
```

This event is triggered before the LoadRows method starts. Return False from the event handler to prevent the rows load operation from occurring.

TDataSet.BeforeLoadColumns Event

```
property BeforeLoadColumns: TDataSetEvent
```

This event is triggered before the LoadColumns method starts. Return False from the event handler to prevent the columns load operation from occurring.

TDataSet.BeforeOpen Event

```
property BeforeOpen: TDataSetEvent
```

This event is triggered before the Open method starts. Return False from the event handler to prevent the open from occurring.

TDataSet.BeforeSave Event

```
property BeforeSave: TDataSetEvent
```

This event is triggered before the Save method starts. Return False from the event handler to prevent the save from occurring.

TDataSet.BeforeScroll Event

```
property BeforeScroll: TDataSetEvent
```

This event is triggered before any navigation operation starts for a dataset. The following TDataSet methods cause the BeforeScroll event to be triggered:

First
Prior
Next
Last
MoveBy
MoveTo
Find
Sort
GotoBookmark

Return False from the event handler to prevent the navigation from occurring.

TDataSet.BeforeUpdate Event

```
property BeforeUpdate: TDataSetEvent
```

This event is triggered before the Update method starts. Return False from the event handler to prevent the update from occurring.

TDataSet.OnCalculateRow Event

```
property OnCalculateRow: TDataRowEvent
```

This event is triggered whenever a column in a row of dataset is updated. You can use this event to re-compute any calculated columns defined in the dataset.

Note
You cannot modify non-calculated columns in an event handler attached to this event.

TDataSet.OnInitRow Event

```
property OnInitRow: TNotifyEvent
```

This event is triggered when the Insert method is called and provides an opportunity to initialize a new row with default values. Any columns modified in this event handler have their Modified property reset after the event handler for this event is done executing.

TDataSet.OnLoadColumnsError Event

```
property OnLoadColumnsError: TDataSetErrorEvent
```

This event is triggered whenever an error occurs during the execution of the LoadColumns method.

Note
 If an event handler is not assigned to this event, then the error will be raised as an exception.

TDataSet.OnLoadError Event

```
property OnLoadError: TDataSetErrorEvent
```

This event is triggered whenever an error occurs during the execution of the LoadRows method.

Note
If an event handler is not assigned to this event, then the error will be raised as an exception.

TDataSet.OnRowChanged Event

```
property OnRowChanged: TDataRowEvent
```

This event is triggered whenever the active row, or any column in the active row, is changed.

TDataSet.OnSortChanged Event

```
property OnSortChanged: TDataRowEvent
```

This event is triggered whenever the active sort is changed.

TDataSet.OnStateChange Event

```
property OnStateChange: TNotifyEvent
```

This event is triggered whenever the State property changes.

10.56 TDataSetController Component

Unit: WebData

Inherits From TComponent

The TDataSetController component provides a way for a developer to respond to changes in a TDataSet instance that is located on the same form/database or another form/database in a similar fashion to data-bound controls. It also allows the developer to initiate the editing of a TDataSet instance via the Edit method. The TDataSet instance to attach to is specified via the DataSet property.

Properties	Methods	Events
DataSet	Edit	OnDataChanged
		OnSortChanged
		OnStateChange

TDataSetController.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset that the controller should attach to.

TDataSetController.Edit Method

```
function Edit: Boolean
```

Use this method to cause the attached DataSet to begin editing the current row. If the dataset is empty, then a new row will be added.

TDataSetController.OnDataChanged Event

```
property OnDataChanged: TDataRowEvent
```

This event is triggered whenever the DataSet is changed via scrolling and navigation, inserts, updates, or deletes.

TDataSetController.OnSortChanged Event

```
property OnSortChanged: TDataRowEvent
```

This event is triggered whenever the DataSet's active sort changes.

TDataSetController.OnStateChange Event

```
property OnStateChange: TNotifyEvent
```

This event is triggered whenever the DataSet's State changes.

10.57 TDataSetToolBar Component

Unit: WebTlbrs

Inherits From TToolBarControl

The TDataSetToolBar class represents a dataset toolbar control. When a dataset toolbar control is bound to a TDataSet instance, it can be used to navigate and update the dataset and its buttons will automatically be enabled/disabled to reflect the current dataset state.

Properties	Methods	Events
Background		OnAnimationComplete
Border		OnAnimationsComplete
Buttons		OnButtonClick
Corners		OnHide
Cursor		OnMove
DataSet		OnShow
Hint		OnSize
InsetShadow		
Opacity		

TDataSetToolBar.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TDataSetToolBar.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TDataSetToolBar.Buttons Property

```
property Buttons: TDataSetToolBarButtons
```

Contains the pre-defined buttons for the toolbar.

TDataSetToolBar.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TDataSetToolBar.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TDataSetToolBar.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TDataSetToolBar.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TDataSetToolBar.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Specifies the inset shadow for the control.

TDataSetToolBar.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TDataSetToolBar.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TDataSetToolBar.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TDataSetToolBar.OnButtonClick Event

```
property OnButtonClick: TClickEvent
```

This event is triggered whenever one of the toolbar buttons is clicked.

TDataSetToolBar.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TDataSetToolBar.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TDataSetToolBar.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TDataSetToolBar.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

10.58 TDataSetToolBarButton Component

Unit: WebTlbrs

Inherits From TToolBarButton

The TDataSetToolBarButton class represents a dataset toolbar button control contained within a TDataSetToolBar instance.

Properties	Methods	Events

10.59 TDataSetToolBarButtons Component

Unit: WebTlbrs

Inherits From TComponent

The TDataSetToolBarButtons class represents a collection of dataset toolbar button controls for the various operations required for a TDataSet instance bound to the container TDataSetToolBar instance. These buttons can be used to navigate and update the dataset and they will automatically be enabled/disabled to reflect the current dataset state.

Properties	Methods	Events
AppendButton		
CancelButton		
DeleteButton		
FindButton		
FirstButton		
LastButton		
NextButton		
PriorButton		
SaveButton		
UpdateButton		

TDataSetToolBarButtons.AppendButton Property

```
property AppendButton: TToolBarButton
```

Specifies the properties of the Append button.

TDataSetToolBarButtons.CancelButton Property

```
property CancelButton: TToolBarButton
```

Specifies the properties of the Cancel button.

TDataSetToolBarButtons.DeleteButton Property

```
property DeleteButton: TToolBarButton
```

Specifies the properties of the Delete button.

TDataSetToolBarButtons.FindButton Property

```
property FindButton: TToolBarButton
```

Specifies the properties of the Find button.

Note
By default, the Find button doesn't actually perform any action when clicked. You need to assign an event handler to the TDataSetToolBar OnButtonClick event in order to perform an action when the button is clicked.

TDataSetToolBarButtons.FirstButton Property

```
property FirstButton: TToolBarButton
```

Specifies the properties of the First button.

TDataSetToolBarButtons.LastButton Property

```
property LastButton: TToolBarButton
```

Specifies the properties of the Last button.

TDataSetToolBarButtons.NextButton Property

```
property NextButton: TToolBarButton
```

Specifies the properties of the Next button.

TDataSetToolBarButtons.PriorButton Property

```
property PriorButton: TToolBarButton
```

Specifies the properties of the Prior button.

TDataSetToolBarButtons.SaveButton Property

```
property SaveButton: TToolBarButton
```

Specifies the properties of the Save button.

TDataSetToolBarButtons.UpdateButton Property

```
property UpdateButton: TToolBarButton
```

Specifies the properties of the Update button.

10.60 TDataValue Component

Unit: WebCore

Inherits From TObject

The TDataValue class represents the value for a column in a row in a TDataSet component. The TDataValue class is an abstract class and is implemented fully in the TStringValue, TBooleanValue, TIntegerValue, TFloatValue, TDateValue, TTimeValue, TDateTimeValue, and TBlobValue classes.

Properties	Methods	Events
AsBoolean	Assign	
AsDate	Clear	
AsDateTime	Compare	
AsFloat	Create	
AsInteger	GetJSON	
AsString		
AsTime		
Modified		
Null		

TDataValue.AsBoolean Property

```
property AsBoolean: Boolean
```

Gets or sets the column value as a boolean value. If the column value cannot be expressed as a boolean value, then an exception will be raised.

TDataValue.AsDate Property

```
property AsDate: DateTime
```

Gets or sets the column value as a date value. If the column value cannot be expressed as a date value, then an exception will be raised.

TDataValue.AsDateTime Property

```
property AsDateTime: DateTime
```

Gets or sets the column value as a date/time value. If the column value cannot be expressed as a date/time value, then an exception will be raised.

TDataValue.AsFloat Property

```
property AsFloat: Double
```

Gets or sets the column value as a floating-point value. If the column value cannot be expressed as a floating-point value, then an exception will be raised.

TDataValue.AsInteger Property

```
property AsInteger: Integer
```

Gets or sets the column value as an integer value. If the column value cannot be expressed as an integer value, then an exception will be raised.

TDataValue.AsString Property

```
property AsString: String
```

Gets or sets the column value as a string value. If the column value cannot be expressed as a string value, then an exception will be raised.

TDataValue.AsTime Property

```
property AsTime: DateTime
```

Gets or sets the column value as a time value. If the column value cannot be expressed as a time value, then an exception will be raised.

TDataValue.Modified Property

```
property Modified: Boolean
```

Indicates whether the column value has been modified. This property will only be True if the TDataSet component that owns the row in which the column value resides is in an insert or update State.

Note
This property also applies to setting a column value to NULL by calling the Clear method.

TDataValue.Null Property

```
property Null: Boolean
```

Indicates that the column value does not contain an actual value and is, instead, NULL.

TDataValue.Assign Method

```
procedure Assign(Value: TDataValue)
```

Use this method to assign the contents of the column value passed as a parameter to the column value. When assigning a column value, the following attributes of the column value are copied:

- Null property
- Modified property
- The column value itself (if Null property is False)

TDataValue.Clear Method

```
procedure Clear
```

Use this method to clear the column value and set the Null property to True. This will also cause the Modified property to be set to True.

TDataValue.Compare Method

```
function Compare(Value: TDataValue; CaseInsensitive:
      Boolean=False; LocaleInsensitive: Boolean=False): Integer
```

Use this method to compare a column value against another column value:

- If the column value parameter is greater than the column value, then -1 will be returned.
- If the column value parameter is less than the column value, then 1 will be returned.
- If the column value parameter is equal to the column value, then 0 will be returned.

The CaseInsensitive parameter only applies to string column values, and determines if the column values are compared in a case-sensitive or case-insensitive manner.

TDataValue.Create Method

```
constructor Create(AValues: TDataValues=nil; AIndex: Integer=-1)
```

Use this method to create a new instance of the TDataValue class. The AValues parameter indicates the data values that will contain the instance and the optional AIndex parameter is used to indicate the position of the value for retrieving schema information used with the data value.

TDataValue.GetJSON Method

```
function GetJSON: String
```

Use this method to retrieve a JSON string that represents the column name and value.

10.61 TDataValues Component

Unit: WebCore

Inherits From TObject

The TDataValues class represents a set of data values, and is used as the base class for the TDataRow class used with datasets.

Properties	Methods	Events
Count	Assign	
ID	Compare	
Modified	Create	
Values	GetJSON	
	Initialize	

TDataValues.Count Property

```
property Count: Integer
```

Indicates the number of values in the set of data values.

TDataValues.ID Property

```
property ID: Integer
```

Indicates the unique ID of the data value set.

TDataValues.Modified Property

```
property Modified: Boolean
```

Indicates whether any of the data values have been modified.

TDataValues.Values Property

```
property Values[Index: Integer]: TDataValue
```

Accesses all data values in the set by index.

TDataValues.Assign Method

```
procedure Assign(NewValues: TDataValues)
```

Use this method to assign the values of all data values in the source set to the current set of data values.

Warning
It is assumed that both sets of data values contain the same number of data values.

TDataValues.Compare Method

```
function Compare(Value: TDataValues; CaseInsensitive:
      Boolean=False; LocaleInsensitive: Boolean=False): Integer
```

Use this method to compares the values of all data values in the source set to the current set of data values.

Warning
It is assumed that both sets of data values contain the same number of data values.

TDataValues.Create Method

```
constructor Create(AID: Integer; ACount: Integer)
```

Use this method to create a new instance of the TDataValues class. The AID parameter indicates the user-defined ID for the instance ACount parameter indicates the number of data values that should be created.

TDataValues.GetJSON Method

```
function GetJSON(ModifiedOnly: Boolean=False; IncludeID:
      Boolean=False; const IDName: String=''): String
```

Use this method to retrieve a JSON string that represents the set of data values as a JSON object containing the data values as properties.

TDataValues.Initialize Method

```
procedure Initialize
```

Use this method to clear all data values so that they are NULL and marked as not modified.

10.62 TDateEditComboBox Component

Unit: WebEdits

Inherits From TEditComboControl

The TDateEditComboBox component represents a date edit combo box control. A date edit combo box is a combo box control that allows the user to directly enter a date value or select a date value from a drop-down calendar.

Properties	Methods	Events
Alignment		OnAnimationComplete
AutoComplete		OnAnimationsComplete
CalendarDefaultView		OnButtonClick
CalendarHeight		OnChange
CalendarWidth		OnClick
Cursor		OnDblClick
DataColumn		OnDropDownHide
DataSet		OnDropDownShow
Direction		OnEnter
DropDownPosition		OnExit
Enabled		OnHide
Font		OnKeyDown
Hint		OnKeyPress
LocalizeText		OnKeyUp
MaxLength		OnMouseDown
ReadOnly		OnMouseEnter
SelectedDate		OnMouseLeave
TabOrder		OnMouseMove
TabStop		OnMouseUp
Text		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TDateEditComboBox.Alignment Property

```
property Alignment: TContentAlignment
```

Specifies the alignment of the input value for the control.

TDateEditComboBox.AutoComplete Property

```
property AutoComplete: TAutoCompleteType
```

Specifies how to handle auto-completion for the control. Auto-completion allows the browser to display a list of suggestions for input values, based upon earlier input values entered by the user.

TDateEditComboBox.CalendarDefaultView Property

```
property CalendarDefaultView: TCalendarView
```

Specifies the default view for the drop-down calendar. The default view determines both the initial view shown in the calendar after it is created, as well as the minimum view that the user is permitted to navigate to. The default value is cvMonth.

TDateEditComboBox.CalendarHeight Property

```
property CalendarHeight: Integer
```

Specifies the height of the drop-down calendar.

TDateEditComboBox.CalendarWidth Property

```
property CalendarWidth: Integer
```

Specifies the width of the drop-down calendar.

TDateEditComboBox.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TDateEditComboBox.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TDateEditComboBox.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TDateEditComboBox.Direction Property

```
property Direction: TContentDirection
```

Specifies the direction in which the text is displayed/edited.

TDateEditComboBox.DropDownPosition Property

```
property DropDownPosition: TDropDownPosition
```

Specifies where the drop-down calendar will be positioned.

TDateEditComboBox.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TDateEditComboBox.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TDateEditComboBox.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TDateEditComboBox.LocalizeText Property

```
property LocalizeText: Boolean
```

Specifies whether date assignments (as strings) to the control's Text property are treated as local dates or UTC dates. The default value is True.

Note

This property only affects the value of the SelectedDate property and does not affect how date values are saved to and from a dataset column via the DataSet and DataColumn properties.

TDateEditComboBox.MaxLength Property

```
property MaxLength: Integer
```

Specifies the maximum allowable length, in characters, of the Text property for the control. A value of 0 specifies an unlimited allowable length.

TDateEditComboBox.ReadOnly Property

```
property ReadOnly: Boolean
```

Specifies whether the control's input value can be modified by the user. The default value is False.

Note
The input value can always be programmatically modified.

TDateEditComboBox.SelectedDate Property

```
property SelectedDate: DateTime
```

Specifies the selected date for the control.

TDateEditComboBox.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TDateEditComboBox.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TDateEditComboBox.Text Property

```
property Text: String
```

Specifies the control's input value as a string.

TDateEditComboBox.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TDateEditComboBox.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TDateEditComboBox.OnButtonClick Event

```
property OnButtonClick: TClickEvent
```

This event is triggered whenever the associated combo button is clicked.

TDateEditComboBox.OnChange Event

```
property OnChange: TNotifyEvent
```

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TDateEditComboBox.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TDateEditComboBox.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TDateEditComboBox.OnDropDownHide Event

```
property OnDropDownHide: TNotifyEvent
```

This event is triggered when the associated drop-down control is hidden.

TDateEditComboBox.OnDropDownShow Event

```
property OnDropDownShow: TNotifyEvent
```

This event is triggered when the associated drop-down control is shown.

TDateEditComboBox.OnEnter Event

```
property OnEnter: TNotifyEvent
```

This event is triggered when the control obtains input focus.

TDateEditComboBox.OnExit Event

```
property OnExit: TNotifyEvent
```

This event is triggered when the control loses input focus.

TDateEditComboBox.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TDateEditComboBox.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when the control has input focus and the user presses a key or key combination.

TDateEditComboBox.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TDateEditComboBox.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when the control has input focus and the user releases a key or key combination.

TDateEditComboBox.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TDateEditComboBox.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TDateEditComboBox.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TDateEditComboBox.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TDateEditComboBox.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TDateEditComboBox.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TDateEditComboBox.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TDateEditComboBox.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TDateEditComboBox.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TDateEditComboBox.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TDateEditComboBox.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TDateEditComboBox.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.63 TDateTimeValue Component

Unit: WebCore

Inherits From TDataValue

This class represents the value for a DateTime column in a row in a TDataSet component.

Properties	Methods	Events

10.64 TDateValue Component

Unit: WebCore

Inherits From TDateTimeValue

This class represents the value for a Date column in a TDataSet component.

Properties	Methods	Events

10.65 TDialog Component

Unit: WebForms

Inherits From TDialogControl

The TDialog component represents a dialog form control with a border and a caption bar with a close button. TDialogButton components can be used with a TDialog instance to provide standard dialog keystroke and button click functionality.

Properties	Methods	Events
ActivateOnClick		OnAnimationComplete
CaptionBar		OnAnimationsComplete
Client		OnCaptionBarDblClick
CloseOnEscape		OnClick
Corners		OnClose
Cursor		OnCloseQuery
Opacity		OnDblClick
OutsetShadow		OnHide
Sizable		OnKeyDown
		OnKeyPress
		OnKeyUp
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMouseWheel
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TDialog.ActivateOnClick Property

```
property ActivateOnClick: Boolean
```

Specifies whether the dialog should automatically be brought to the front when it, or any child controls, are clicked.

Note
 This property only has an effect when the dialog is parented to another control. By default, dialogs always are brought to the front when clicked.

TDialog.CaptionBar Property

```
property CaptionBar: TDialogCaptionBar
```

Specifies the properties of the caption bar for the control.

TDialog.Client Property

```
property Client: TDialogClient
```

Specifies the properties of the client area for the control.

TDialog.CloseOnEscape Property

```
property CloseOnEscape: Boolean
```

Specifies whether the dialog is automatically closed when the user hits the Escape key. The default value is True.

> **Note**
> If there is a TDialogButton instance on the dialog with its ModalCancel property set to True, then the button will be clicked when the escape key is pressed and this property will ignored.

TDialog.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TDialog.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TDialog.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TDialog.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the control.

TDialog.Sizable Property

```
property Sizable: Boolean
```

Specifies whether the dialog should be sizable. When a dialog is sizable, the user will be able to click on the bottom-right border of the dialog to begin sizing the dialog interactively, and moving the mouse while holding down the left mouse button will allow the user to size the dialog according to their needs.

TDialog.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TDialog.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TDialog.OnCaptionBarDblClick Event

```
property OnCaptionBarDblClick: TNotifyEvent
```

This event is triggered when the caption bar is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TDialog.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TDialog.OnClose Event

```
property OnClose: TNotifyEvent
```

This event is triggered when the dialog is closed by the user via the caption bar close button, or when the Close method is called.

TDialog.OnCloseQuery Event

```
property OnCloseQuery: TCloseQueryEvent
```

This event is triggered when the dialog is closed by the user via the caption bar close button, or when the Close method is called.

Return True to allow the close to continue, or False to prevent the dialog from closing.

TDialog.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TDialog.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TDialog.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when a child control has input focus and the user presses a key or key combination.

TDialog.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when a child control has input focus and presses/releases a key or key combination.

TDialog.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when a child control has input focus and the user releases a key or key combination.

TDialog.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TDialog.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TDialog.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TDialog.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TDialog.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TDialog.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

This event is triggered whenever the mouse wheel is rotated forward or backward.

TDialog.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TDialog.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TDialog.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TDialog.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TDialog.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TDialog.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TDialog.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.66 TDialogButton Component

Unit: WebBtns

Inherits From TButton

The TDialogButton component represents a dialog button control. A dialog button control allows the user to initiate a specific action by using a mouse click or by pushing the spacebar or enter key. In addition, the ModalResult property can be used when a dialog button control is placed on a modal form. If the ModalResult property is set, then clicking the dialog button control will automatically cause the form to close and return the same modal result in the ModalResult property for the form.

Properties	Methods	Events
ModalCancel		
ModalDefault		
ModalResult		

TDialogButton.ModalCancel Property

```
property ModalCancel: Boolean
```

Specifies that this button is the cancel button for the parent dialog. If the escape key is pressed while the dialog is active, then the ModalResult property for the dialog will be set to the same value as the ModalResult property of the button.

TDialogButton.ModalDefault Property

```
property ModalDefault: Boolean
```

Specifies that this button is the default button for the parent dialog. If the enter key is pressed while the dialog is active, then the ModalResult property for the dialog will be set to the same value as the ModalResult property of the button.

TDialogButton.ModalResult Property

```
property ModalResult: TModalResult
```

Specifies the result to assign to a form's ModalResult property when the button is clicked.

10.67 TDialogCaptionBar Component

Unit: WebForms

Inherits From TCaptionBarControl

The TDialogCaptionBar component represents the caption bar for a TDialog form control, and includes a caption and a close button.

Properties	Methods	Events
Alignment		
AllowClose		
AllowMove		
Background		
Caption		
Cursor		
Font		
Icon		
Padding		

TDialogCaptionBar.Alignment Property

```
property Alignment: TContentAlignment
```

Specifies the alignment of the caption in the caption bar.

TDialogCaptionBar.AllowClose Property

```
property AllowClose: Boolean
```

Specifies whether the close button should be shown in the caption bar.

TDialogCaptionBar.AllowMove Property

```
property AllowMove: Boolean
```

Specifies whether the user can press and hold a mouse or touch on the caption bar and drag the container dialog to a new position.

TDialogCaptionBar.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TDialogCaptionBar.Caption Property

```
property Caption: String
```

Specifies the caption to display in the caption bar.

TDialogCaptionBar.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TDialogCaptionBar.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TDialogCaptionBar.Icon Property

```
property Icon: TIconProperties
```

Specifies the properties of the icon used with the caption bar.

TDialogCaptionBar.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

10.68 TDialogClient Component

Unit: WebForms

Inherits From TComponent

The TDialogClient component represents the client area for a TDialog form control.

Properties	Methods	Events
Background		
InsetShadow		
Padding		

TDialogClient.Background Property

```
property Background: TBackground
```

Specifies the background for the control.

TDialogClient.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Specifies the inset shadow for the control.

TDialogClient.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

10.69 TDialogControl Component

Unit: WebForms

Inherits From TFormControl

The TDialogControl control is the base class for dialogs, and contains all of the core dialog functionality in the form of public methods and protected properties/events that descendant classes can use to create customized dialogs.

Properties	Methods	Events

10.70 TDialogEditComboBox Component

Unit: WebEdits

Inherits From TEditComboControl

The TDialogEditComboBox component represents a dialog edit combo box control. A dialog edit combo box is a combo box control that allows the user to directly enter an input value or trigger the display of a custom selection dialog by intercepting the combo button clicks.

Properties	Methods	Events
Alignment		OnAnimationComplete
AutoComplete		OnAnimationsComplete
Cursor		OnButtonClick
DataColumn		OnChange
DataSet		OnClick
Direction		OnDblClick
Enabled		OnEnter
Font		OnExit
Hint		OnHide
MaxLength		OnKeyDown
ReadOnly		OnKeyPress
SpellCheck		OnKeyUp
TabOrder		OnMouseDown
TabStop		OnMouseEnter
Text		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TDialogEditComboBox.Alignment Property

```
property Alignment: TContentAlignment
```

Specifies the alignment of the input value for the control.

TDialogEditComboBox.AutoComplete Property

```
property AutoComplete: TAutoCompleteType
```

Specifies how to handle auto-completion for the control. Auto-completion allows the browser to display a list of suggestions for input values, based upon earlier input values entered by the user.

TDialogEditComboBox.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TDialogEditComboBox.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TDialogEditComboBox.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TDialogEditComboBox.Direction Property

```
property Direction: TContentDirection
```

Specifies the direction in which the text is displayed/edited.

TDialogEditComboBox.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TDialogEditComboBox.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TDialogEditComboBox.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TDialogEditComboBox.MaxLength Property

```
property MaxLength: Integer
```

Specifies the maximum allowable length, in characters, of the Text property for the control. A value of 0 specifies an unlimited allowable length.

TDialogEditComboBox.ReadOnly Property

```
property ReadOnly: Boolean
```

Specifies whether the control's input value can be modified by the user. The default value is False.

Note
The input value can always be programmatically modified.

TDialogEditComboBox.SpellCheck Property

```
property SpellCheck: Boolean
```

Specifies whether spell-checking will be enabled for the control.

TDialogEditComboBox.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TDialogEditComboBox.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TDialogEditComboBox.Text Property

```
property Text: String
```

Specifies the control's input value as a string.

TDialogEditComboBox.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TDialogEditComboBox.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TDialogEditComboBox.OnButtonClick Event

```
property OnButtonClick: TClickEvent
```

This event is triggered whenever the associated combo button is clicked.

TDialogEditComboBox.OnChange Event

```
property OnChange: TNotifyEvent
```

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TDialogEditComboBox.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TDialogEditComboBox.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TDialogEditComboBox.OnEnter Event

```
property OnEnter: TNotifyEvent
```

This event is triggered when the control obtains input focus.

TDialogEditComboBox.OnExit Event

```
property OnExit: TNotifyEvent
```

This event is triggered when the control loses input focus.

TDialogEditComboBox.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TDialogEditComboBox.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when the control has input focus and the user presses a key or key combination.

TDialogEditComboBox.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TDialogEditComboBox.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when the control has input focus and the user releases a key or key combination.

TDialogEditComboBox.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TDialogEditComboBox.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TDialogEditComboBox.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TDialogEditComboBox.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TDialogEditComboBox.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TDialogEditComboBox.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TDialogEditComboBox.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TDialogEditComboBox.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TDialogEditComboBox.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TDialogEditComboBox.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TDialogEditComboBox.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TDialogEditComboBox.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.71 TDivElement Component

Unit: WebUI

Inherits From TElement

The TDivElement class is the default element class for UI elements, and contains all of the element functionality in the form of public methods and properties/events that control classes can use to create any type of control.

> **Note**
> The TDivElement is just a typecast of the TElement class, and is only used for associating an element class type with a specific type of run-time HTML tag (<div>).

Properties	Methods	Events

10.72 TDropDownButtonControl Component

Unit: WebEdits

Inherits From TButtonInputControl

The TDropDownButtonControl control is the base class for drop-down button controls, and contains all of the core drop-down functionality in the form of public methods and protected properties/events that descendant classes can use to create customized drop-down button controls.

Properties	Methods	Events
	HideDropDown	
	ShowDropDown	

TDropDownButtonControl.HideDropDown Method

```
procedure HideDropDown
```

Use this method to hide the drop-down control associated with the control (if visible).

TDropDownButtonControl.ShowDropDown Method

```
procedure ShowDropDown
```

Use this method to show the drop-down control associated with the control (if not already visible).

10.73 TDropDownEditControl Component

Unit: WebEdits

Inherits From TEditControl

The TDropDownEditControl control is the base class for drop-down edit controls, and contains all of the core drop-down functionality in the form of public methods and protected properties/events that descendant classes can use to create customized drop-down edit controls.

Properties	Methods	Events
	HideDropDown	
	ShowDropDown	

TDropDownEditControl.HideDropDown Method

```
procedure HideDropDown
```

Use this method to hide the drop-down control associated with the control (if visible).

TDropDownEditControl.ShowDropDown Method

```
procedure ShowDropDown
```

Use this method to show the drop-down control associated with the control (if not already visible).

10.74 TEdit Component

Unit: WebEdits

Inherits From TEditControl

The TEdit component represents an edit control. An edit control allows the user to directly enter an input value using the keyboard.

Properties	Methods	Events
Alignment		OnAnimationComplete
AutoComplete		OnAnimationsComplete
Cursor		OnChange
DataColumn		OnClick
DataSet		OnDblClick
Direction		OnEnter
Enabled		OnExit
Font		OnHide
Hint		OnKeyDown
InputType		OnKeyPress
MaxLength		OnKeyUp
ReadOnly		OnMouseDown
SpellCheck		OnMouseEnter
TabOrder		OnMouseLeave
TabStop		OnMouseMove
Text		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TEdit.Alignment Property

```
property Alignment: TContentAlignment
```

Specifies the alignment of the input value for the control.

TEdit.AutoComplete Property

```
property AutoComplete: TAutoCompleteType
```

Specifies how to handle auto-completion for the control. Auto-completion allows the browser to display a list of suggestions for input values, based upon earlier input values entered by the user.

TEdit.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TEdit.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TEdit.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TEdit.Direction Property

```
property Direction: TContentDirection
```

Specifies the direction in which the text is displayed/edited.

TEdit.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TEdit.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TEdit.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TEdit.InputType Property

```
property InputType: TTextInputType
```

Specifies the type of text being input into the element by the user. This information is used by the browser to determine how to display and edit the text. For example, in touch environments, this property is used to determine which soft keyboard to display to the user.

TEdit.MaxLength Property

```
property MaxLength: Integer
```

Specifies the maximum allowable length, in characters, of the Text property for the control. A value of 0 specifies an unlimited allowable length.

TEdit.ReadOnly Property

```
property ReadOnly: Boolean
```

Specifies whether the control's input value can be modified by the user. The default value is False.

> **Note**
> The input value can always be programmatically modified.

TEdit.SpellCheck Property

```
property SpellCheck: Boolean
```

Specifies whether spell-checking will be enabled for the control.

TEdit.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TEdit.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TEdit.Text Property

```
property Text: String
```

Specifies the control's input value as a string.

TEdit.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TEdit.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TEdit.OnChange Event

```
property OnChange: TNotifyEvent
```

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TEdit.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TEdit.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TEdit.OnEnter Event

```
property OnEnter: TNotifyEvent
```

This event is triggered when the control obtains input focus.

TEdit.OnExit Event

```
property OnExit: TNotifyEvent
```

This event is triggered when the control loses input focus.

TEdit.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TEdit.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when the control has input focus and the user presses a key or key combination.

TEdit.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TEdit.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when the control has input focus and the user releases a key or key combination.

TEdit.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TEdit.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TEdit.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TEdit.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TEdit.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TEdit.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TEdit.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TEdit.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TEdit.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TEdit.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TEdit.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TEdit.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.75 TEditComboBox Component

Unit: WebEdits

Inherits From TEditComboControl

The TEditComboBox component represents an edit combo box control. An edit combo box is a combo control that allows the user to directly enter an input value or select an input value from a drop-down list of values.

> **Note**
> If you do not want to allow for direct editing of the input value, please consider using a TButtonComboBox component instead.

Properties	Methods	Events
Alignment		OnButtonClick
AutoComplete		OnChange
AutoDropDown		OnClick
AutoItemHeight		OnDblClick
Cursor		OnDropDownHide
DataColumn		OnDropDownShow
DataSet		OnEnter
Direction		OnExit
DropDownItemCount		OnHide
DropDownPosition		OnKeyDown
Enabled		OnKeyPress
Font		OnKeyUp
Hint		OnMouseDown
ItemHeight		OnMouseEnter
ItemIndex		OnMouseLeave
Items		OnMouseMove
KeyPressInterval		OnMouseUp
MaxLength		OnMove
ReadOnly		OnShow
Sorted		OnSize
SpellCheck		OnTouchCancel
TabOrder		OnTouchEnd
TabStop		OnTouchMove
Text		OnTouchStart

TEditComboBox.Alignment Property

```
property Alignment: TContentAlignment
```

Specifies the alignment of the input value for the control.

TEditComboBox.AutoComplete Property

```
property AutoComplete: TAutoCompleteType
```

Specifies how to handle auto-completion for the control. Auto-completion allows the browser to display a list of suggestions for input values, based upon earlier input values entered by the user.

TEditComboBox.AutoDropDown Property

```
property AutoDropDown: Boolean
```

Specifies that the drop-down list of Items should automatically be shown when the user starts typing. The default value is False.

TEditComboBox.AutoItemHeight Property

```
property AutoItemHeight: Boolean
```

Specifies that the displayed height of the drop-down items will automatically be set based upon the Font property settings. The default value is True.

TEditComboBox.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TEditComboBox.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TEditComboBox.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TEditComboBox.Direction Property

```
property Direction: TContentDirection
```

Specifies the direction in which the text is displayed/edited.

TEditComboBox.DropDownItemCount Property

```
property DropDownItemCount: Integer
```

Specifies the number of visible items to display in the drop-down list of Items.

TEditComboBox.DropDownPosition Property

```
property DropDownPosition: TDropDownPosition
```

Specifies where the drop-down list will be positioned.

TEditComboBox.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TEditComboBox.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TEditComboBox.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TEditComboBox.ItemHeight Property

```
property ItemHeight: Integer
```

Specifies the height, in pixels, of the items displayed in the drop-down list.

TEditComboBox.ItemIndex Property

```
property ItemIndex: Integer
```

Specifies the index of the selected item in the drop-down list of Items, or -1 if there is no selected item.

TEditComboBox.Items Property

```
property Items: TStrings
```

Specifies the items to use for the drop-down list.

TEditComboBox.KeyPressInterval Property

```
property KeyPressInterval: Integer
```

Specifies the interval, in milliseconds, that is used by the control to combine user keystrokes into a search value that is then used for performing a near search on the Items property. Effectively, this means that the user has KeyPressInterval milliseconds in which to hit a key in order for the keystroke to be included as part of a near search. The default value is 300 milliseconds.

For example, if the user hits the 'S', 'M', and 'I' keys within the KeyPressInterval property value, but hits the 'T' key outside of the KeyPressInterval property, then the control will perform a near search using the value 'SMI', followed by a near search using the value 'T'.

TEditComboBox.MaxLength Property

```
property MaxLength: Integer
```

Specifies the maximum allowable length, in characters, of the Text property for the control. A value of 0 specifies an unlimited allowable length.

TEditComboBox.ReadOnly Property

```
property ReadOnly: Boolean
```

Specifies whether the control's input value can be modified by the user. The default value is False.

Note
The input value can always be programmatically modified.

TEditComboBox.Sorted Property

```
property Sorted: Boolean
```

Specifies whether the drop-down items will automatically be sorted. The default value is False.

TEditComboBox.SpellCheck Property

```
property SpellCheck: Boolean
```

Specifies whether spell-checking will be enabled for the control.

TEditComboBox.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TEditComboBox.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TEditComboBox.Text Property

```
property Text: String
```

Specifies the control's input value as a string.

TEditComboBox.OnButtonClick Event

```
property OnButtonClick: TClickEvent
```

This event is triggered whenever the associated combo button is clicked.

TEditComboBox.OnChange Event

```
property OnChange: TNotifyEvent
```

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TEditComboBox.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TEditComboBox.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TEditComboBox.OnDropDownHide Event

```
property OnDropDownHide: TNotifyEvent
```

This event is triggered when the associated drop-down control is hidden.

TEditComboBox.OnDropDownShow Event

```
property OnDropDownShow: TNotifyEvent
```

This event is triggered when the associated drop-down control is shown.

TEditComboBox.OnEnter Event

```
property OnEnter: TNotifyEvent
```

This event is triggered when the control obtains input focus.

TEditComboBox.OnExit Event

```
property OnExit: TNotifyEvent
```

This event is triggered when the control loses input focus.

TEditComboBox.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TEditComboBox.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when the control has input focus and the user presses a key or key combination.

TEditComboBox.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TEditComboBox.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when the control has input focus and the user releases a key or key combination.

TEditComboBox.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TEditComboBox.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TEditComboBox.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TEditComboBox.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TEditComboBox.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TEditComboBox.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TEditComboBox.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TEditComboBox.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TEditComboBox.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TEditComboBox.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TEditComboBox.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TEditComboBox.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.76 TEditComboControl Component

Unit: WebEdits

Inherits From TDropDownEditControl

The TEditComboControl control is the base class for edit combo controls, and contains all of the core combo functionality in the form of public methods and protected properties/events that descendant classes can use to create customized edit combo controls.

Properties	Methods	Events

10.77 TEditControl Component

Unit: WebEdits

Inherits From TInputControl

The TEditControl control is the base class for text edit controls, and contains all of the core text edit functionality in the form of public methods and protected properties/events that descendant classes can use to create customized text edit controls.

Properties	Methods	Events
SelectionEnd	SelectAll	
SelectionStart	SelectNone	

TEditControl.SelectionEnd Property

```
property SelectionEnd: Integer
```

Specifies the ending position (0-based) of the selected characters in the text. For example, if the control contains the text "Hello World", setting the SelectionStart property to 6 and the SelectionEnd property to 11 will result in the text "World" being selected.

TEditControl.SelectionStart Property

```
property SelectionStart: Integer
```

Specifies the starting position (0-based) of the selected characters in the text. For example, if the control contains the text "Hello World", setting the SelectionStart property to 6 and the SelectionEnd property to 11 will result in the text "World" being selected.

TEditControl.SelectAll Method

```
procedure SelectAll
```

Use this method to select all of the text content in the edit control.

TEditControl.SelectNone Method

```
procedure SelectNone
```

Use this method to unselect any previously-selected text content in the edit control.

10.78 TElement Component

Unit: WebUI

Inherits From TPersistent

The TElement class is the base element class for all UI elements, and contains all of the element functionality in the form of public methods and properties/events that control classes can use to create any type of control.

Properties	Methods	Events
AltKey	Assign	
AlwaysOnTop	BeginRotation	
Animations	BeginUpdate	
ApplyProperties	BringToFront	
AutoSize	CancelRotation	
Background	CanTab	
Border	Create	
Button	DefineLayout	
ClientHeight	EndUpdate	
ClientWidth	FindByName	
ComposedValue	FindByPos	
Constraints	FindFirstTab	
Content	FindLastTab	
Controller	FindNextTab	
ControllerElement	FindPriorTab	
Corners	ForceUpdate	
CtrlKey	GetTabCount	
Cursor	Hide	
DefinedHeight	IsControllerClass	
DefinedLeft	IsParentOf	
DefinedTop	Minimize	
DefinedWidth	RemoveFocus	
Delegate	Restore	
DisplayIndex	ScrollBy	
DOMElement	SendToBack	

Elements	SetFocus	
Enabled	Show	
EventX		
EventY		
FocusEnabled		
Font		
FontSizeAnimation		
ForceDefaultCursor		
Format		
HasElements		
Height		
Hint		
HTMLContentEnabled		
ID		
InsetShadow		
InUpdate		
IsVisible		
KeyChar		
KeyCode		
Layout		
LayoutIndex		
Left		
Margins		
Minimized		
Name		
Opacity		
OutsetShadow		
OverflowX		
OverflowY		
Padding		
Parent		
PreserveFocus		
ScrollEnabled		
ScrollHeight		

ScrollLeft		
ScrollTop		
ScrollWidth		
ShiftKey		
TabEnabled		
TabIndex		
TagName		
Top		
TotalHeight		
TotalWidth		
Visible		
WheelDelta		
Width		

TElement.AltKey Property

```
property AltKey: Boolean
```

Indicates whether the Alt key was pressed during the last triggered event for this element.

TElement.AlwaysOnTop Property

```
property AlwaysOnTop: Boolean
```

Specifies that the element should always be visually placed on top of any other element within the same container element.

TElement.Animations Property

```
property Animations: TAnimations
```

Specifies the animations for the element.

TElement.ApplyProperties Property

```
property ApplyProperties: TElementProperties
```

Specifies which properies of the element to apply to a UI element when applying the state of a control interface to a control.

TElement.AutoSize Property

```
property AutoSize: TAutoSize
```

Specifies how (if at all) the element should automatically be sized based upon the Content and Format properties.

TElement.Background Property

```
property Background: TBackground
```

Specifies the background of the element.

TElement.Border Property

```
property Border: TBorder
```

Specifies the border of the element.

TElement.Button Property

```
property Button: Integer
```

Indicates the code of the mouse button for the last triggered mouse event for this element.

TElement.ClientHeight Property

```
property ClientHeight: Integer
```

Indicates the height of the client rectangle for the element.

TElement.ClientWidth Property

```
property ClientWidth: Integer
```

Indicates the width of the client rectangle for the element.

TElement.ComposedValue Property

```
property ComposedValue: String
```

Indicates the current composed value after the last triggered composition event for this element.

TElement.Constraints Property

```
property Constraints: TConstraints
```

Indicates the dimensional constraints for the element.

TElement.Content Property

```
property Content: String
```

Specifies the content of the element. The Font and Format properties control how the content is displayed.

Note
At this time, Elevate Web Builder only supports the use of plain text as content, but this will expand to include HTML content in a future release.

TElement.Controller Property

```
property Controller: TInterfaceController
```

Specifies the controller instance for the element. A controller instance is assigned to any element that serves as the base element for a TInterfaceController class descendant.

> **Note**
> With the Elevate Web Builder component library, the TInterfaceController class is never instantiated directly. It is used only as a bridge component between the TElement class and the TControl class.

TElement.ControllerElement Property

```
property ControllerElement: TElement
```

Indicates the element, relative to the current element instance, that has its Controller property assigned. If the current element instance has its Controller property assigned, then this property will be equal to the current element instance.

TElement.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical corner radii for the element.

TElement.CtrlKey Property

```
property CtrlKey: Boolean
```

Indicates whether the Control key was pressed during the last triggered event for this element.

TElement.Cursor Property

```
property Cursor: TCursor
```

Specifies the type of cursor to show for the element.

Note
The ForceDefaultCursor property is useful for situations where the type of cursor is set to crAuto and the browser would normally show the cursor as a text caret for text selection, which is the case with any elements containing text as content.

TElement.DefinedHeight Property

```
property DefinedHeight: Integer
```

Indicates the **defined** height for the element. The defined height is the last value that was directly assigned to the Height. Layout property changes may affect the value returned by the Height property, so this property is useful when you don't want the calculated height of an element.

TElement.DefinedLeft Property

```
property DefinedLeft: Integer
```

Indicates the **defined** left position for the element. The defined left position is the last value that was directly assigned to the Left. Layout property changes may affect the value returned by the Left property, so this property is useful when you don't want the calculated left position of an element.

TElement.DefinedTop Property

```
property DefinedTop: Integer
```

Indicates the **defined** top position for the element. The defined top position is the last value that was directly assigned to the Top. Layout property changes may affect the value returned by the Top property, so this property is useful when you don't want the calculated top position of an element.

TElement.DefinedWidth Property

```
property DefinedWidth: Integer
```

Indicates the **defined** width for the element. The defined width is the last value that was directly assigned to the Width. Layout property changes may affect the value returned by the Height property, so this property is useful when you don't want the calculated width of an element.

TElement.Delegate Property

```
property Delegate: TElement
```

Specifies that any focus/tabbing for the element should be delegated to the specified element. The specified element should be a child element of the current element instance. Delegation is useful for situations where a child element is the element that should be focused when the current element instance is focused.

TElement.DisplayIndex Property

```
property DisplayIndex: Integer
```

Specifies the display index, or visual stacking index, of the element within its parent container element.

TElement.DOMElement Property

```
property DOMElement: THTMLElement
```

Contains a reference to the underlying DOM element associated with the element.

TElement.Elements Property

```
property Elements: TElements
```

Contains references to all child elements that have the current element instance assigned as its Parent property. Element parentage automatically implies element ownership in Elevate Web Builder, so any child elements of the current element instance will automatically be disposed of when the current element instance is destroyed.

Warning
This property will be nil if no child elements have been assigned to the element.

TElement.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the element is enabled or disabled.

TElement.EventX Property

```
property EventX: Integer
```

Indicates the horizontal position of the mouse button/touch for the last triggered mouse/touch event for this element.

TElement.EventY Property

```
property EventY: Integer
```

Indicates the vertical position of the mouse button/touch for the last triggered mouse/touch event for this element.

TElement.FocusEnabled Property

```
property FocusEnabled: Boolean
```

Specifies whether the element should be focusable.

TElement.Font Property

```
property Font: TFont
```

Specifies font to use for the element content.

TElement.FontSizeAnimation Property

```
property FontSizeAnimation: TAnimation
```

Specifies the animation properties for the element font's Size property.

TElement.ForceDefaultCursor Property

```
property ForceDefaultCursor: Boolean
```

Specifies that the cursor should be dynamically changed to crDefault if the Cursor property is set to crAuto and the browser would normally show the cursor as a text caret for text selection, which is the case with any elements containing text as content.

TElement.Format Property

```
property Format: TFormat
```

Specifies the content formatting to use for the element's content.

TElement.HasElements Property

```
property HasElements: Boolean
```

Indicates whether the Elements is nil, and if not, whether the there are any child elements in the list of elements.

TElement.Height Property

```
property Height: Integer
```

Indicates the current height of the element, and can be used to specify the **defined** height for the element. The defined height is the last value that was directly assigned to the Height property. Layout property changes may affect the value returned by the Height property.

TElement.Hint Property

```
property Hint: String
```

Specifies a popup hint to display during a mouse-over in a desktop browser or a touch in a mobile browser.

TElement.HTMLContentEnabled Property

```
property HTMLContentEnabled: Boolean
```

Specifies whether the element can contain HTML content. If this property is False, then any text assigned to the Content property will treated as plain text. If this property is True, then any text assigned to the Content property will be treated as HTML.

> **Note**
> Enabling this property modifies how the UI layout functionality treats the content with respect to measurement and display.

TElement.ID Property

```
property ID: String
```

Specifies a unique DOM ID for the element.

TElement.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Specifies the inset shadow for the element. The inset shadow appears within the bounds of the client rectangle for the element.

TElement.InUpdate Property

```
property InUpdate: Boolean
```

Indicates whether the element, or any of its parent elements, is currently in a batch update. An element is in a batch update if the BeginUpdate method is called on the element or any of its parent elements. When an element is in a batch update, it doesn't apply any changes to any of its properties until the EndUpdate method is called on the element or any of its parent elements, and the InUpdate property returns False.

> **Note**
> Updates are reference-counted so calls to the BeginUpdate method increment the reference count, and calls to the EndUpdate method decrement the reference count.

TElement.IsVisible Property

```
property IsVisible: Boolean
```

Indicates whether the element, and **all** of its parent elements, are visible. This is in contrast to the Visible property, which only indicates whether the current element is visible.

TElement.KeyChar Property

```
property KeyChar: Char
```

Indicates the character that represents the keystroke for the last triggered keypress event for this element.

TElement.KeyCode Property

```
property KeyCode: Integer
```

Indicates the code of the keystroke for the last triggered key event for this element.

TElement.Layout Property

```
property Layout: TLayout
```

Specifies the layout for the element.

TElement.LayoutIndex Property

```
property LayoutIndex: Integer
```

Specifies the layout index of the element. The layout index determines the order in which the child elements of a container element are positioned and sized using the layout management functionality for UI elements.

TElement.Left Property

```
property Left: Integer
```

Indicates the current left position of the element, and can be used to specify the **defined** left position for the element. The defined left position is the last value that was directly assigned to the Left property. Layout property changes may affect the value returned by the Left property.

TElement.Margins Property

```
property Margins: TMargins
```

Specifies the margins to be used for the element when the element is being positioned/sized using the layout management functionality.

TElement.Minimized Property

```
property Minimized: Boolean
```

Indicates whether the element is currently minimized.

TElement.Name Property

```
property Name: String
```

Specifies the name of the element. Element names must be unique within a an interface state and within the base and child elements of a visual control.

TElement.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the element, from 0 (transparent) to 100 (opaque).

TElement.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the element. The outset shadow appears behind the bounds of the element.

TElement.OverflowX Property

```
property OverflowX: TOverflowType
```

Specifies whether or not to show a native horizontal browser scrollbar for the element if the width of its contents and/or child elements exceeds the width of the client rectangle for the element.

Note

This property should be left at its default value of otHidden for most elements. Elevate Web Builder only uses this element property for specifying the scrollbars of the application viewport.

TElement.OverflowY Property

```
property OverflowY: TOverflowType
```

Specifies whether or not to show a native vertical browser scrollbar for the element if the height of its contents and/or child elements exceeds the height of the client rectangle for the element.

Note
This property should be left at its default value of otHidden for most elements. Elevate Web Builder only uses this element property for specifying the scrollbars of the application viewport.

TElement.Padding Property

```
property Padding: TPadding
```

Specifies any padding for the element. The padding of an element shrinks the width and height of the client rectangle for the element.

TElement.Parent Property

```
property Parent: TElement
```

Specifies the parent element for the element. The only element whose Parent property is always nil is the interface manager's root element.

TElement.PreserveFocus Property

```
property PreserveFocus: Boolean
```

Specifies whether the element should preserve focus on an existing focused element if an attempt is made to focus the element and the FocusEnabled property is False. This property is used by non-focusable elements such as scrollbar control elements to make sure that clicking on the elements does not cause any currently-focused elements to lose focus.

TElement.ScrollEnabled Property

```
property ScrollEnabled: Boolean
```

Specifies whether the element wants mouse wheel and touch scroll events routed to it. Whenever a mouse wheel or touch scroll event occurs, the event manager will automatically route such events to the first container element that has its ScrollEnabled property set to True.

TElement.ScrollHeight Property

```
property ScrollHeight: Integer
```

Indicates the total height of the element's content and/or its child elements. If an element's content height is greater than its Height property, then you can use the ScrollTop property to programmatically scroll the element vertically, or to find out the current vertical scroll position.

TElement.ScrollLeft Property

```
property ScrollLeft: Integer
```

Specifies the horizontal scroll position for the element. If an element's ScrollWidth property is greater than its Width property, then you can use this property to programmatically scroll the element horizontally, or to find out the current horizontal scroll position.

TElement.ScrollTop Property

```
property ScrollTop: Integer
```

Specifies the vertical scroll position for the element. If an element's ScrollHeight property is greater than its Height property, then you can use this property to programmatically scroll the element vertically, or to determine the current vertical scroll position.

TElement.ScrollWidth Property

```
property ScrollWidth: Integer
```

Indicates the total width of the element's content and/or its child elements. If an element's content width is greater than its Width property, then you can use the ScrollLeft property to programmatically scroll the element horizontally, or to determine the current horizontal scroll position.

TElement.ShiftKey Property

```
property ShiftKey: Boolean
```

Indicates whether the Shift key was pressed during the last triggered event for this element.

TElement.TabEnabled Property

```
property TabEnabled: Boolean
```

Specifies whether the element will take part in any tabbing order within its parent container element.

TElement.TabIndex Property

```
property TabIndex: Integer
```

Specifies the tab index of the element within its parent container element.

TElement.TagName Property

```
property TagName: String
```

Indicates the HTML tag name associated with the element at runtime. HTML tag names are used at runtime to create elements using a mapping between the tag name and a given TElement class type.

TElement.Top Property

```
property Top: Integer
```

Indicates the current top position of the element, and can be used to specify the **defined** top position for the element. The defined top position is the last value that was directly assigned to the Top property. Layout property changes may affect the value returned by the Top property.

TElement.TotalHeight Property

```
property TotalHeight: Integer
```

Indicates the total height of the element, including its top and bottom margins.

TElement.TotalWidth Property

```
property TotalWidth: Integer
```

Indicates the total width of the element, including its left and right margins.

TElement.Visible Property

```
property Visible: Boolean
```

Specifies whether the element is visible or not.

TElement.WheelDelta Property

```
property WheelDelta: Integer
```

Indicates the mouse wheel delta for the last triggered mouse wheel event for this element.

TElement.Width Property

```
property Width: Integer
```

Indicates the current width of the element, and can be used to specify the **defined** width for the element. The defined width is the last value that was directly assigned to the Width property. Layout property changes may affect the value returned by the Width property.

TElement.Assign Method

```
procedure Assign(AElement: TElement)
```

Use this method to assign all of the properties of a source element to the element.

Note
This method will also recursively assign all child elements from the source element to the element, so please be careful when using this method.

TElement.BeginRotation Method

```
procedure BeginRotation(AStyle: TAnimationStyle; AInterval:
      Integer)
```

Use this method to begin rotating the contents of the element using the animation properties specified by the AStyle and AInterval parameters. The rotation animation will continue until the CancelRotation method is called.

Note
The Elevate Web Builder component library uses this functionality for animating the rotation of font icons in progress dialogs.

TElement.BeginUpdate Method

```
procedure BeginUpdate
```

Use this method to begin a batch update on an element. An element is in a batch update if the BeginUpdate method is called on the element, or any of its parent elements. When an element is in a batch update, it doesn't apply any changes to any of its properties until the EndUpdate method is called on the element or any of its parent elements, and the InUpdate property returns False.

Note
Updates are reference-counted so calls to the BeginUpdate method increment the reference count, and calls to the EndUpdate method decrement the reference count.

TElement.BringToFront Method

```
procedure BringToFront
```

Use this method to bring the element to the front of the visual stacking order. An element in the front will have a DisplayIndex property equal to one less than the number of child elements in the parent container element.

TElement.CancelRotation Method

```
procedure CancelRotation
```

Use this method to cancel the rotation animation started using the BeginRotation method.

Note
The Elevate Web Builder component library uses this functionality for animating the rotation of font icons in progress dialogs.

TElement.CanTab Method

```
function CanTab(AClass: TInterfaceControllerClass=nil): Boolean
```

Use this method to determine if an element, with or without a particular associated Controller instance, is visible, enabled, and can be focused, or contains a child element that can be focused.

TElement.Create Method

```
constructor Create(const AName: String; AParent: TElement=nil;
      const ATagName: String='')
```

Use this method to create a new instance of the TElement class. The AName parameter indicates the name of the element, the optional AParent parameter indicates the parent of the element, and the optional ATagName parameter indicates the HTML tag name to associate with the element. The HTML tag name is used to create the underlying browser DOM element that will be managed by the element at runtime.

TElement.DefineLayout Method

```
procedure DefineLayout
```

Use this method to assign the current bounds of the element to the Left, Top, Width, and Height properties of the element.

This method is useful in situations where an element has been positioned or sized according to its Layout properties, but you wish to have the bounds persist even after modifying the layout properties so that they no longer position or size the element in the same way.

TElement.EndUpdate Method

```
procedure EndUpdate
```

Use this method to end a batch update on an element. An element is in a batch update if the BeginUpdate method is called on the element or any of its parent elements. When an element is in a batch update, it doesn't apply any changes to any of its properties until the EndUpdate method is called on the element or any of its parent elements, and the InUpdate property returns False.

Note

Updates are reference-counted so calls to the BeginUpdate method increment the reference count, and calls to the EndUpdate method decrement the reference count.

TElement.FindByName Method

```
function FindByName(const AName: String): TElement
```

Use this method to find an element with a specified name. The search for the element includes the current element instance.

Note
This is a recursive method, and will search through grandchildren, great-grandchildren, etc. for an element with the specified name.

TElement.FindByPos Method

```
function FindByPos(X,Y: Integer): TElement
```

TElement.FindFirstTab Method

```
function FindFirstTab(AClass: TInterfaceControllerClass=nil):
      TElement
```

Use this method to find the first child element in TabIndex order, with or without a particular associated Controller instance, whose CanTab method returns True.

TElement.FindLastTab Method

```
function FindLastTab(AClass: TInterfaceControllerClass=nil):
      TElement
```

Use this method to find the last child element in TabIndex order, with or without a particular associated Controller instance, whose CanTab method returns True.

TElement.FindNextTab Method

```
function FindNextTab(AElement: TElement; AWrap: Boolean=False;
      AClass: TInterfaceControllerClass=nil): TElement
```

Use this method to find the next child element in TabIndex order, with or without a particular associated Controller instance, whose CanTab method returns True.

TElement.FindPriorTab Method

```
function FindPriorTab(AElement: TElement; AWrap: Boolean=False;
      AClass: TInterfaceControllerClass=nil): TElement
```

Use this method to find the prior child element in TabIndex order, with or without a particular associated Controller instance, whose CanTab method returns True.

TElement.ForceUpdate Method

```
procedure ForceUpdate
```

Use this method to force the element to perform a layout update when in the middle of a BeginUpdate/EndUpdate block. Normally, an element will not update its layout bounds when an update block is in effect. A forced layout update will ensure that the element's layout bounds are updated to reflect any layout changes applied to the element within the update block.

TElement.GetTabCount Method

```
function GetTabCount(AClass: TInterfaceControllerClass=nil):
      Integer
```

Use this method to get the total number of child elements, with or without a particular associated Controller instance, whose CanTab method returns True.

Note
This is a recursive method, and will return the total number of **all** child elements, even grandchildren, great-grandchildren, etc.

TElement.Hide Method

```
procedure Hide
```

Use this method to set the Visible property to False and hide the element.

TElement.IsControllerClass Method

```
function IsControllerClass(AClass: TInterfaceControllerClass;
      ANameRequired: Boolean=False): Boolean
```

Use this method to determine if the element's associated Controller is the specified controller class type.

TElement.IsParentOf Method

```
function IsParentOf(AElement: TElement): Boolean
```

Use this method to determine if the element is the parent of the specified element.

Note
This is a recursive method, and will search **all** child elements, even grandchildren, great-grandchildren, etc. to determine if any of the child elements are the parent of the specified element.

TElement.Minimize Method

```
function Minimize: Boolean
```

Use this method to minimize the element. Minimizing an element saves the current width and height of the element so that it can be restored later using the Restore method, and sets the Minimized to True.

This method will return True if the element was not already minimized and False if it was already minimized.

TElement.RemoveFocus Method

```
procedure RemoveFocus
```

Use this method to remove focus from the element. If the element is not focused, then this method does nothing.

TElement.Restore Method

```
function Restore: Boolean
```

Use this method to restore a minimized element. A minimized element is an element whose Minimized property is True.

This method will return True if the element was minimized and False if it was not minimized.

TElement.ScrollBy Method

```
procedure ScrollBy(X,Y: Integer)
```

If an element's content width and/or height is greater than its Width and Height properties, then you can use this method to scroll the contents of the element horizontally, vertically, or both. The X and Y values represent the number of pixels to scroll the contents by, and may be negative values for scrolling backward.

TElement.SendToBack Method

```
procedure SendToBack
```

Use this method to send the element to the back of the visual stacking order. An element in the back will have a DisplayIndex property equal to 0.

TElement.SetFocus Method

```
procedure SetFocus
```

Use this method to set focus to the element. If the element's FocusEnabled property is set to True, then the element will be focused. If the element's PreserveFocus property is set to False, then focus will be removed from whatever element is currently focused.

TElement.Show Method

```
procedure Show
```

Use this method to set the Visible to True and show the element.

10.79 TElementAttribute Component

Unit: WebUI

Inherits From TPersistent

The TElementAttribute class represents an attribute for a UI element or control. It is the base class for all attribute classes like the TBackground, TFont, and TLayout classes and contains the base functionality for loading and change management for element attributes.

Properties	Methods	Events
	Assign	
	Create	
	GetStyle	

TElementAttribute.Assign Method

```
procedure Assign(AAttribute: TElementAttribute)
```

Use this method to assign the properties of a source element attribute to the element attribute.

TElementAttribute.Create Method

```
constructor Create(AElement: TElement; AParent:
      TElementAttribute)
```

Use this method to create a new instance of the TElementAttribute class. The AElement parameter indicates the element instance that will manage the attribute, and the AParent parameter indicates the parent attribute, if any, that the attribute is contained within. The parent attribute is used to aggregate change management at the outermost attribute so as to avoid excessively triggering change notifications in the element.

TElementAttribute.GetStyle Method

```
function GetStyle: String
```

Use this method at run-time to get a string containing the CSS style data for the element attribute.

Note
This method is not available at design-time for element attributes.

10.80 TElementProperties Component

Unit: WebUI

Inherits From TPersistent

The TElementProperties class specifies which properties of an element are applied when a control interface state is applied to the elements of a control instance.

Properties	Methods	Events
AlwaysOnTop		
AutoSize		
Background		
Border		
Constraints		
Content		
Corners		
Cursor		
DisplayIndex		
Font		
FontColor		
FontSize		
FontStyle		
Format		
Height		
InsetShadow		
Layout		
LayoutIndex		
Left		
Margins		
Opacity		
OutsetShadow		
Padding		
Top		
Visible		
Width		

TElementProperties.AlwaysOnTop Property

```
property AlwaysOnTop: Boolean
```

Specifies that the AlwaysOnTop property should be applied.

TElementProperties.AutoSize Property

```
property AutoSize: Boolean
```

Specifies that the AutoSize property should be applied.

TElementProperties.Background Property

```
property Background: Boolean
```

Specifies that the Background property should be applied.

TElementProperties.Border Property

```
property Border: Boolean
```

Specifies that the Border property should be applied.

TElementProperties.Constraints Property

```
property Constraints: Boolean
```

Specifies that the Constraints property should be applied.

TElementProperties.Content Property

```
property Content: Boolean
```

Specifies that the Content property should be applied.

TElementProperties.Corners Property

```
property Corners: Boolean
```

Specifies that the Corners property should be applied.

TElementProperties.Cursor Property

```
property Cursor: Boolean
```

Specifies that the Cursor property should be applied.

TElementProperties.DisplayIndex Property

```
property DisplayIndex: Boolean
```

Specifies that the DisplayIndex property should be applied.

TElementProperties.Font Property

```
property Font: Boolean
```

Specifies that the Font property should be applied.

TElementProperties.FontColor Property

```
property FontColor: Boolean
```

Specifies that the Font.Color property should be applied.

TElementProperties.FontSize Property

```
property FontSize: Boolean
```

Specifies that the Font.Size property should be applied.

TElementProperties.FontStyle Property

```
property FontStyle: Boolean
```

Specifies that the Font.Style property should be applied.

TElementProperties.Format Property

```
property Format: Boolean
```

Specifies that the Format property should be applied.

TElementProperties.Height Property

```
property Height: Boolean
```

Specifies that the Height property should be applied.

TElementProperties.InsetShadow Property

```
property InsetShadow: Boolean
```

Specifies that the InsetShadow property should be applied.

TElementProperties.Layout Property

```
property Layout: Boolean
```

Specifies that the Layout property should be applied.

TElementProperties.LayoutIndex Property

```
property LayoutIndex: Boolean
```

Specifies that the LayoutIndex property should be applied.

TElementProperties.Left Property

```
property Left: Boolean
```

Specifies that the Left property should be applied.

TElementProperties.Margins Property

```
property Margins: Boolean
```

Specifies that the Margins property should be applied.

TElementProperties.Opacity Property

```
property Opacity: Boolean
```

Specifies that the Opacity property should be applied.

TElementProperties.OutsetShadow Property

```
property OutsetShadow: Boolean
```

Specifies that the OutsetShadow property should be applied.

TElementProperties.Padding Property

```
property Padding: Boolean
```

Specifies that the Padding property should be applied.

TElementProperties.Top Property

```
property Top: Boolean
```

Specifies that the Top property should be applied.

TElementProperties.Visible Property

```
property Visible: Boolean
```

Specifies that the Visible property should be applied.

TElementProperties.Width Property

```
property Width: Boolean
```

Specifies that the Width property should be applied.

10.81 TElements Component

Unit: WebUI

Inherits From TPersistent

The TElements class is a container for the child elements of an element.

Properties	Methods	Events
Count	Assign	
DisplayElement	Create	
Element		
TabCount		
TabElement		

TElements.Count Property

```
property Count: Integer
```

Indicates the number of child elements.

TElements.DisplayElement Property

```
property DisplayElement[Index: Integer]: TElement
```

Accesses child elements by their DisplayIndex property.

TElements.Element Property

```
property Element[Index: Integer]: TElement
```

Accesses child elements by their LayoutIndex property.

TElements.TabCount Property

```
property TabCount: Integer
```

Indicates the number of focusable and tabable child elements.

TElements.TabElement Property

```
property TabElement[Index: Integer]: TElement
```

Accesses child elements by their TabIndex property.

TElements.Assign Method

```
procedure Assign(AElements: TElements)
```

Use this method to assign a source list of child elements to the elements list.

Note
This method will also recursively assign all child elements of each source child element, so please be careful when using this method.

TElements.Create Method

```
constructor Create(AElement: TElement)
```

Use this method to create a new instance of the TElements class. The AElement parameter indicates the parent element instance that will manage the elements.

10.82 TFileComboBox Component

Unit: WebEdits

Inherits From TEditComboControl

The TFileComboBox component represents a file combo box control. A file combo box is a combo box control that allows the user to select a file from the browser's file selection dialog.

Properties	Methods	Events
AcceptTypes		OnAnimationComplete
Cursor		OnAnimationsComplete
Enabled		OnButtonClick
Font		OnChange
Hint		OnClick
ReadOnly		OnDblClick
TabOrder		OnEnter
TabStop		OnExit
Text		OnHide
		OnKeyDown
		OnKeyPress
		OnKeyUp
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TFileComboBox.AcceptTypes Property

```
property AcceptTypes: TStrings
```

Specifies a list of MIME types or file extensions for filtering the list of files that are shown to the user when the file combo box control's drop-down button is clicked.

TFileComboBox.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TFileComboBox.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TFileComboBox.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TFileComboBox.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TFileComboBox.ReadOnly Property

```
property ReadOnly: Boolean
```

Specifies whether the control's input value can be modified by the user. The default value is False.

Note
The input value can always be programmatically modified.

TFileComboBox.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TFileComboBox.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TFileComboBox.Text Property

```
property Text: String
```

Specifies the control's input value as a string.

TFileComboBox.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TFileComboBox.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TFileComboBox.OnButtonClick Event

```
property OnButtonClick: TClickEvent
```

This event is triggered whenever the associated combo button is clicked.

TFileComboBox.OnChange Event

```
property OnChange: TNotifyEvent
```

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TFileComboBox.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TFileComboBox.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TFileComboBox.OnEnter Event

```
property OnEnter: TNotifyEvent
```

This event is triggered when the control obtains input focus.

TFileComboBox.OnExit Event

```
property OnExit: TNotifyEvent
```

This event is triggered when the control loses input focus.

TFileComboBox.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TFileComboBox.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when the control has input focus and the user presses a key or key combination.

TFileComboBox.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TFileComboBox.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when the control has input focus and the user releases a key or key combination.

TFileComboBox.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TFileComboBox.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TFileComboBox.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TFileComboBox.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TFileComboBox.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TFileComboBox.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TFileComboBox.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TFileComboBox.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TFileComboBox.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TFileComboBox.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TFileComboBox.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TFileComboBox.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.83 TFileInputElement Component

Unit: WebUI

Inherits From TInputElement

The TFileInputElement class is the element class for file input elements, and contains all of the file input functionality in the form of public methods and properties/events that control classes can use to create file upload controls.

> **Note**
> This element does not provide support for file uploads at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events
AcceptTypes	Click	

TFileInputElement.AcceptTypes Property

```
property AcceptTypes: String
```

Specifies a comma-delimited list of MIME types or file extensions for filtering the list of files that are shown to the user when the file input element is clicked.

TFileInputElement.Click Method

```
procedure Click
```

Use this method to programmatically simulate a click on the element.

Note
Due to security restrictions, an application can only successfully call this method if the calling code was initiated by a user interfaction, such as a mouse click or touch.

10.84 TFill Component

Unit: WebUI

Inherits From TElementAttribute

The TFill class represents the background fill of a UI element or control. Background fills can be solid colors (including transparent) or gradients.

Properties	Methods	Events
Color	SetToDefault	
FillType		
Gradient		

TFill.Color Property

```
property Color: TColor
```

Specifies the color of the background fill.

TFill.FillType Property

```
property FillType: TFillType
```

Specifies the type of background fill.

TFill.Gradient Property

```
property Gradient: TGradient
```

Specifies a gradient background fill.

TFill.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the background fill's properties to their default values.

10.85 TFloatValue Component

Unit: WebCore

Inherits From TDataValue

This class represents the value for a Float column in a TDataSet component.

Properties	Methods	Events

10.86 TFont Component

Unit: WebUI

Inherits From TElementAttribute

The TFont class represents the font to use for the content of a UI element or control.

Properties	Methods	Events
Color	SetToDefault	
GenericFamily		
LineHeight		
Name		
Size		
Style		

TFont.Color Property

```
property Color: TColor
```

Specifies the color of the font. The default value is clBlack.

TFont.GenericFamily Property

```
property GenericFamily: TGenericFontFamily
```

Specifies the generic font family for the font.

> **Note**
> This property is automatically populated at design-time when the Name property is changed.

TFont.LineHeight Property

```
property LineHeight: Integer
```

Indicates the calculated line height, in pixels, based upon the Size property.

TFont.Name Property

```
property Name: String
```

Specifies the name of the font. The default value is 'Arial'.

TFont.Size Property

```
property Size: Integer
```

Specifies the size, in pixels, of the font. The default value is 16.

TFont.Style Property

```
property Style: TFontStyle
```

Specifies the style of the font.

TFont.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the font's properties to their default values.

10.87 TFontIcon Component

Unit: WebCtrls

Inherits From TComponent

The TFontIcon component represents the font icon for a TIcon control. It includes properties that control the attributes of the font icon, such as the size and color of the icon.

Properties	Methods	Events
AutoSize		
Color		
Size		

TFontIcon.AutoSize Property

```
property AutoSize: Boolean
```

Specifies that the font icon should automatically be sized based upon the TIconProperties Height property.

TFontIcon.Color Property

```
property Color: TColor
```

Specifies the color of the font icon. This property defaults to the pre-defined font icon color of the icon specified by the IconName property, and may change when the IconName property is changed.

TFontIcon.Size Property

```
property Size: Integer
```

Specifies the size, in pixels, of the font icon. This property defaults to the pre-defined font icon color of the icon specified by the IconName property, and may change when the IconName property is changed.

10.88 TFontStyle Component

Unit: WebUI

Inherits From TElementAttribute

The TFontStyle class represents the style attributes of the font to use for the content of a UI element or control.

Properties	Methods	Events
Bold	SetToDefault	
Italic		
Strikeout		
Underline		

TFontStyle.Bold Property

```
property Bold: Boolean
```

Specifies that the font should be bold. The default value is False.

TFontStyle.Italic Property

```
property Italic: Boolean
```

Specifies that the font should be italicized. The default value is False.

TFontStyle.Strikeout Property

```
property Strikeout: Boolean
```

Specifies that the font should have a line drawn through it. The default value is False.

TFontStyle.Underline Property

```
property Underline: Boolean
```

Specifies that the font should be underlined. The default value is False.

TFontStyle.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the font style's properties to their default values.

10.89 TForm Component

Unit: WebForms

Inherits From TFormControl

The TForm component represents the basic form control used in Elevate Web Builder. Please see the Creating and Showing Forms for more information on using forms.

Properties	Methods	Events
ActivateOnClick		OnAnimationComplete
Background		OnAnimationsComplete
Border		OnClick
Corners		OnClose
Cursor		OnCloseQuery
InsetShadow		OnDblClick
Opacity		OnHide
OutsetShadow		OnKeyDown
Padding		OnKeyPress
ScrollBars		OnKeyUp
ScrollSupport		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMouseWheel
		OnMove
		OnScroll
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchScroll
		OnTouchStart

TForm.ActivateOnClick Property

```
property ActivateOnClick: Boolean
```

Specifies whether the form should automatically be brought to the front when it, or any child controls, are clicked.

Note
This property only has an effect when the form is parented to another control. By default, forms always are brought to the front when clicked.

TForm.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TForm.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TForm.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TForm.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TForm.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Specifies the inset shadow for the control.

TForm.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TForm.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the control.

TForm.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

TForm.ScrollBars Property

```
property ScrollBars: TScrollBars
```

Specifies which scrollbars to show, if any.

> **Note**
> Even if this property is set to sbHorizontal, sbVertical, or sbBoth, a scrollbar will only be shown if the size of the contents and/or the child controls of the control exceed the client rectangle for the control.

TForm.ScrollSupport Property

```
property ScrollSupport: TScrollSupport
```

Specifies the directions in which the control can be scrolled, if any.

> **Note**
> This property only applies to scrolling via touch and mouse wheel movements, and does not apply to programmatic scrolling or scrolling via the scroll bars themselves.

TForm.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TForm.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TForm.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TForm.OnClose Event

```
property OnClose: TNotifyEvent
```

This event is triggered when the form's Close method is called.

TForm.OnCloseQuery Event

```
property OnCloseQuery: TCloseQueryEvent
```

This event is triggered when the form's Close method is called.

Return True to allow the close to continue, or False to prevent the form from closing.

TForm.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TForm.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TForm.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when a child control has input focus and the user presses a key or key combination.

TForm.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when a child control has input focus and presses/releases a key or key combination.

TForm.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when a child control has input focus and the user releases a key or key combination.

TForm.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TForm.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TForm.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TForm.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TForm.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TForm.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

This event is triggered whenever the mouse wheel is rotated forward or backward.

TForm.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TForm.OnScroll Event

```
property OnScroll: TNotifyEvent
```

This event is triggered whenever a scrollable control is scrolled horizontally or vertically.

TForm.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TForm.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TForm.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TForm.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TForm.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TForm.OnTouchScroll Event

```
property OnTouchScroll: TTouchScrollEvent
```

This event is triggered whenever a touch moves in any direction over a touch-scroll-enabled control.

TForm.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.90 TFormat Component

Unit: WebUI

Inherits From TElementAttribute

The TFormat class represents the formatting attributes to use for the content of a UI element or control.

Properties	Methods	Events
Alignment	SetToDefault	
Direction		
Wrap		

TFormat.Alignment Property

```
property Alignment: TContentAlignment
```

Specifies the alignment of the content.

TFormat.Direction Property

```
property Direction: TContentDirection
```

Specifies the direction in which the content is displayed.

TFormat.Wrap Property

```
property Wrap: Boolean
```

Specifies whether the content should be word-wrapped.

TFormat.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the format's properties to their default values.

10.91 TFormatSettings Component

Unit: WebCore

Inherits From TObject

The TFormatSettings class represents the formatting settings for numeric, date, and time literals. An instance of the TFormatSettings class called **FormatSettings** is automatically created by the component library at application startup, so further instances of the TFormatSettings class should not be created.

Properties	Methods	Events
DateSeparator	Create	
DecimalSeparator		
LongDayNames		
LongMonthNames		
ShortDateFormat		
ShortDateFormatComp		
ShortDayNames		
ShortMonthNames		
ShortTimeFormat		
ShortTimeFormatComp		
StartOfWeek		
TimeAMString		
TimePMString		
TimeSeparator		
Translations		
TwoDigitYearCenturyWindow		

TFormatSettings.DateSeparator Property

```
property DateSeparator: Char
```

Specifies the character used to separate the various components of a date literal. The default value of this property is the forward slash (/).

Warning
When specifying a ShortDateFormat that uses a different date separator character, please make sure that you modify the DateSeparator property **before** setting the new ShortDateFormat value.

TFormatSettings.DecimalSeparator Property

```
property DecimalSeparator: Char
```

Specifies the character used to separate the integer portion from the fractional portion in a numeric literal. The default value of this property is the period (.).

TFormatSettings.LongDayNames Property

```
property LongDayNames[Day: Integer]: String
```

Specifies the full day names for all days of the week. The values are indexed by the day, with Monday being the first at index 1 and Sunday being the last at index 7. The default values are Monday, Tuesday, Wednesday, Thursday, Friday, Saturday, and Sunday.

TFormatSettings.LongMonthNames Property

```
property LongMonthNames[Month: Integer]: String
```

Specifies the full month names for all months of the year. The values are indexed by the month, with January being the first at index 1 and December being the last at index 12. The default values are January, February, March, April, May, June, July, August, September, October, November, and December.

TFormatSettings.ShortDateFormat Property

```
property ShortDateFormat: String
```

Specifies the format string used for date literals. The default value of this property is 'M/d/yyyy'.

The following date format specifiers are supported:

Format Specifier	Description
M	The month number with no leading zero
MM	The month number with a leading zero if the month number is less than 10
d	The day number with no leading zero
dd	The day number with a leading zero if the day number is less than 10
yy	The last two digits of the year number with a leading zero (see the TwoDigitYearCenturyWindow property)
yyyy	The full four digits of the year number

Note
For date/time literals, the ShortDateFormat is used in conjunction with the ShortTimeFormat property, with a space separating the two.

Warning
When specifying a ShortDateFormat that uses a different date separator character, please make sure that you modify the DateSeparator property **before** setting the new ShortDateFormat value.

TFormatSettings.ShortDateFormatComp Property

```
property ShortDateFormatComp[Index: Integer]: String
```

Accesses a specific component of the short date format by its index in the defined components of the format.

TFormatSettings.ShortDayNames Property

```
property ShortDayNames[Day: Integer]: String
```

Specifies the abbreviated day names for all days of the week. The values are indexed by the day, with Monday being the first at index 1 and Sunday being the last at index 7. The default values are Mon, Tue, Wed, Thu, Fri, Sat, and Sun.

TFormatSettings.ShortMonthNames Property

```
property ShortMonthNames[Month: Integer]: String
```

Specifies the abbreviated month names for all months of the year. The values are indexed by the month, with January being the first at index 1 and December being the last at index 12. The default values are Jan, Feb, Mar, Apr, May, Jun, Jul, Aug, Sep, Oct, Nov, and Dec.

TFormatSettings.ShortTimeFormat Property

```
property ShortTimeFormat: String
```

Specifies the format string used for time literals. The default value of this property is 'hh:mm tt'.

The following date format specifiers are supported:

Format Specifier	Description
h	The hour number (12-hour clock) with no leading zero
hh	The hour number (12-hour clock) with a leading zero if the hour number is less than 10
H	The hour number (24-hour clock) with no leading zero
HH	The hour number (24-hour clock) with a leading zero if the hour number is less than 10
m	The minute number with no leading zero
mm	The minute number with a leading zero if the minute number is less than 10
s	The second number with no leading zero
ss	The second number with a leading zero if the second number is less than 10
tt	The AM/PM designation for a 12-hour clock literal

Note
For date/time literals, the ShortDateFormat is used in conjunction with the ShortTimeFormat property, with a space separating the two.

Warning
When specifying a ShortTimeFormat that uses a different time separator character, please make sure that you modify the TimeSeparator property **before** setting the new ShortTimeFormat value.

TFormatSettings.ShortTimeFormatComp Property

```
property ShortTimeFormatComp[Index: Integer]: String
```

Accesses a specific component of the short time format by its index in the defined components of the format.

TFormatSettings.StartOfWeek Property

```
property StartOfWeek: Integer
```

Specifies the week day number designated as the start of the week for calendar display purposes. The default value is 7, for Sunday, and the valid values are 1 (Monday) through 7 (Sunday).

TFormatSettings.TimeAMString Property

```
property TimeAMString: String
```

Specifies the literal used to represent the AM designation for a 12-hour clock time literal. The default value of this property is 'AM'.

TFormatSettings.TimePMString Property

```
property TimePMString: String
```

Specifies the literal used to represent the PM designation for a 12-hour clock time literal. The default value of this property is 'PM'.

TFormatSettings.TimeSeparator Property

```
property TimeSeparator: Char
```

Specifies the character used to separate the various components of a time literal. The default value of this property is the colon (:).

Warning
When specifying a ShortTimeFormat that uses a different time separator character, please make sure that you modify the TimeSeparator property **before** setting the new ShortTimeFormat value.

TFormatSettings.Translations Property

```
property Translations: TStrings
```

Specifies the translations for all strings used in the Elevate Web Builder framework for visual controls and error/warning/information messages as key-value pairs. In order to translate one or more strings, simply reference the string ID using the Values property and assign it a new value. For example, the following will translate the text used for the OK button on message dialogs to a new value:

```
FormatSettings.Translations.Values['DLG_BTN_OK']:='Okey-Dokey';
```

The following strings are pre-defined in the framework and can be translated to suit your needs:

String	Default Value
TYPE_UNKNOWN	Unknown
TYPE_STRING	String
TYPE_BOOLEAN	Boolean
TYPE_INTEGER	Integer
TYPE_FLOAT	Float
TYPE_DATE	Date
TYPE_TIME	Time
TYPE_DATETIME	DateTime
TYPE_BLOB	Blob
TYPE_SYMBOL	Symbol
ERR_CMP_DESTROY	"%s" component already destroyed
ERR_LOAD_PERSISTENT	Persistent load error (%s)
ERR_LOAD_METHOD	Method %s not found
ERR_SAVE_PERSISTENT	Persistent save error (%s)
ERR_SET_RANGE	Value "%s" out of range for set
ERR_BOOLEAN_LITERAL	Invalid boolean literal "%s" specified
ERR_FORMAT	Error in the format string %s (%s)
ERR_DATETIME_DATE	date
ERR_DATETIME_TIME	time
ERR_DATETIME_MONTH	Invalid month %s specified
ERR_DATETIME_DAY	Invalid day %s specified

ERR_DATETIME_TOOMANYCOMPS	Too many %s components
ERR_DATETIME_MISSINGCOMP	Missing %s component
ERR_DATETIME_INVALIDCOMP	Invalid %s component
ERR_DATETIME_INVALIDFORMAT	Invalid %s format
ERR_DATETIME_INVALID	Invalid %s (%s)
ERR_PARSE_TERMSTR	Unterminated string at %s
ERR_PARSE_MISSING	Missing %s
ERR_PARSE_EXPECT	Expected %s, instead found %s at %s
ERR_LIST_BOUNDS	List index %s out of bounds
ERR_LIST_SORT	You can only use the Find method when the string list is sorted
ERR_OWNER	Invalid owner class %s passed to constructor
ERR_LOADUI_ELEMENT	Error loading interface element (%s)
ERR_LOADUI_STATE	Error loading interface state (%s)
ERR_UI_ELEMENTCLASS	Cannot find registered element class information for the %s class
ERR_APP_ERRORLINE	Line: %s
ERR_APP_ERRORTITLE	Application Error
ERR_ZOOM_FACTOR	Zoom factor %s invalid, factor must be between 1 and 100
ERR_SLIDE_COUNT	At least %s slide images must be specifed before the slide show can be started
ERR_DOM_EVENTADD	Cannot add event handler to "%s" element for "%s" event
ERR_DOM_EVENTCLEAR	Cannot remove "%s" event handler from "%s"
ERR_HTTP_REQUEST	Error executing request "%s" (%s)
ERR_DATA_DUPCOL	The "%s" column already exists
ERR_DATA_COLNAME	Column names cannot be blank (%s)
ERR_DATA_COLLENGTH	Invalid "%s" column length %s
ERR_DATA_COLSCALE	Invalid "%s" column scale %s
ERR_DATA_CONVERT	Error converting %s value to %s value
ERR_DATA_CONNECT	Cannot connect to server
ERR_DATA_LOADCODE	Status code %s
ERR_DATA_LOAD	Dataset load response error: %s
ERR_DATA_COMMIT	Database commit response error: %s
ERR_DATA_TRANSACTIVE	A transaction is not active
ERR_DATA_PENDREQUEST	There are no pending requests

ERR_DATA_COLUMNS	At least one column must be defined for the "%s" dataset
ERR_DATA_OPEN	The "%s" dataset must be open in order to complete this operation
ERR_DATA_NOTOPEN	The "%s" dataset cannot be open when completing this operation
ERR_DATA_NOTEDITING	The "%s" dataset must be in an editable mode before a column can be assigned a value
ERR_DATA_TRANSCLOSE	Cannot close the "%s" dataset while there are still active transaction operations for the dataset
ERR_DATA_FINDMODE	The "%s" dataset is not in Find mode
ERR_DATA_FINDNEAR	You can only search for nearest matches in the "%s" dataset when searching on columns that match the current sort order
ERR_DATA_COLNOTFOUND	Column "%s" not found
ERR_DATA_TRANSDATASET	Invalid dataset "%s" specified in the transaction operations
ERR_DATA_TRANSOPTYPE	Invalid or unknown operation type %s specified in the transaction operations
ERR_CALENDAR_COLINDEX	Column index %s out of bounds
ERR_CALENDAR_ROWINDEX	Row index %s out of bounds
ERR_GRID_COLINDEX	Column index %s out of bounds
ERR_GRID_ROWINDEX	Row index %s out of bounds
ERR_GRID_COLNOTFOUND	Column "%s" not found
ERR_IMAGE_LOAD	Image "%s" not loaded
ERR_CANVAS	Your browser does not have HTML5 canvas support
ERR_STORAGE	Your browser does not have HTML5 persistent storage support
ERR_SCRIPT_LOAD	Your browser does not support dynamic script loading
ERR_MEDIA	Your browser does not have HTML5 media support
ERR_MAP	The map API has not been loaded
ERR_MAP_GEOCODE	Geocoding request error "%s"
ERR_MAP_LOCNOTFOUND	Location "%s" not found
ERR_MAP_DUPLOC	The "%s" location already exists
ERR_MAP_LOCNAME	Location names cannot be blank (%s)
ERR_DLG_BUTTONS	You must specify at least one button for the message dialog
ERR_FORM_SHOWMODAL	You cannot call ShowModal for the embedded form %s
ERR_APP_SIZERCONTROL	The sizer control itself cannot be assigned as the target control
APP_LOAD_MESSAGE	Loading %s...

ERR_DLG_BUTTONS	You must specify at least one button for the message dialog
DLG_MSG	Message
DLG_BTN_OK	OK
DLG_BTN_CANCEL	Cancel
DLG_BTN_ABORT	Abort
DLG_BTN_RETRY	Retry
DLG_BTN_IGNORE	Ignore
DLG_BTN_YES	Yes
DLG_BTN_NO	No
DLG_BTN_ALL	All
DLG_BTN_NOTOALL	No to All
DLG_BTN_YESTOALL	Yes to All
DLG_BTN_CLOSE	Close

Note
Be sure to include the %s string placeholders in the same order and quantity to ensure that data that is formatted into the string by the framework still produces a string that makes sense.

TFormatSettings.TwoDigitYearCenturyWindow Property

```
property TwoDigitYearCenturyWindow: Integer
```

Specifies what century is added to two-digit years when date literals are converted to DateTime values. This property is subtracted from the current year to determine the pivot year. Two digit years that are prior to the pivot year are interpreted as falling in the next century. The default value is 50.

TFormatSettings.Create Method

```
constructor Create
```

Use this method to create a new instance of the TFormatSettings class.

10.92 TFormControl Component

Unit: WebForms

Inherits From TScrollableControl

The TFormControl control is the base class for forms, and contains all of the core form functionality in the form of public methods and protected properties/events that descendant classes can use to create customized forms.

Properties	Methods	Events
ModalResult	Close	OnCreate
	ShowModal	OnDestroy

TFormControl.ModalResult Property

```
property ModalResult: TModalResult
```

Specifies the result for a modal form. Assigning a value other than mrNone to this property for a modal form will cause the form to attempt to close. This property has no effect upon forms that were not shown modally.

TFormControl.Close Method

```
procedure Close
```

Use this method to close the form. When this method is called, the OnCloseQuery event is triggered, followed by the OnClose event. If the OnCloseQuery event handler returns True, then the form will be hidden before the OnClose event is triggered.

TFormControl.ShowModal Method

```
procedure ShowModal
```

Use this method to show a form in a modal fashion. When a form is shown modally, a modal overlay is placed over all other forms and the entire application surface, and keyboard and mouse input is only allowed for the modal form.

TFormControl.OnCreate Event

```
property OnCreate: TNotifyEvent
```

This event is triggered after the form is created and initialized.

Note
 Any design-time components placed on the form will already be instantiated and initialized before this event is triggered.

TFormControl.OnDestroy Event

```
property OnDestroy: TNotifyEvent
```

This event is triggered before the form is destroyed. Use this event to dispose of any instances or resources that may have been allocated in the OnCreate event handler.

10.93 TFormElement Component

Unit: WebUI

Inherits From TElement

The TFormElement class is the element class for HTML form elements, and contains all of the HTML form functionality in the form of public methods and properties/events that control classes can use to create HTML form controls.

> **Note**
> This element does not provide support for HTML forms at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events
Action	Reset	
Encoding	Submit	
Method		
Target		

TFormElement.Action Property

```
property Action: String
```

Specifies the URL to use for the form submittal when the Submit method is called. The default value is ''.

TFormElement.Encoding Property

```
property Encoding: THTMLFormEncoding
```

Specifies the MIME encoding type of the form data that is sent when the Submit method is called.

TFormElement.Method Property

```
property Method: THTMLFormMethod
```

Specifies the HTTP method used for the form submittal when the Submit method is called.

TFormElement.Target Property

```
property Target: String
```

Specifies the name of a TFrameElement that should accept all output from the form submittal request to the web server when the Submit method is called. The default value is ''.

TFormElement.Reset Method

```
procedure Reset
```

Use this method to clear all form input elements contained within the form element.

TFormElement.Submit Method

```
procedure Submit
```

Use this method to submit all form input elements contained within the form element to the web server using the URL specified by the Action property, the MIME type encoding specified by the Encoding property, and the HTTP method specifed by the Method property.

10.94 TFrameElement Component

Unit: WebUI

Inherits From TWebElement

The TFrameElement class is the element class for embedded iframe elements, and contains all of the iframe functionality in the form of public methods and properties/events that control classes can use to create iframe controls.

> **Note**
> This element does not provide support for iframes at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events
Document	Print	
DocumentText		
Scrolling		
TargetName		
Window		

TFrameElement.Document Property

```
property Document: THTMLDocument
```

Returns the DOM (Document Object Model) document instance of the currently-loaded HTML document. If the URL property has been specified, then this property will return the document instance once the OnLoad event has been triggered and the Loaded property is True.

Note
Accessing the DOM document instance allows you to manipulate the children of the DOM document instance in code instead of having to use HTML strings, which is the case when using the DocumentText property. However, this access is subject to same-origin security constraints, and will be denied if the contents of the frame element were loaded from a different origin.

TFrameElement.DocumentText Property

```
property DocumentText: String
```

Returns the currently-loaded HTML document in the element as a string. If the URL property has been specified, then this property will return the document contents once the OnLoad event has been triggered and the Loaded property is True.

Note

You can also assign a valid HTML string to this property, in which case the URL property is automatically cleared.

TFrameElement.Scrolling Property

```
property Scrolling: Boolean
```

Specifies whether scrolling should be enabled for the element.

Note
This property is required because certain browsers require a special way of specifying whether scrollbars should be shown for an embedded iframe element.

TFrameElement.TargetName Property

```
property TargetName: String
```

Specifies the name of the embedded iframe element, which is required in order to allow output of actions like HTML form submittals to be redirected to a specific iframe element.

TFrameElement.Window Property

```
property Window: TWindow
```

Returns the DOM (Document Object Model) window instance of the frame element.

Note
Accessing the DOM window instance allows you to make calls into the global execution environment of the frame element. However, this access is subject to same-origin security constraints, and will be denied if the contents of the frame element were loaded from a different origin.

TFrameElement.Print Method

```
procedure Print
```

Use this method to print the currently-loaded HTML document in the element. If no HTML document is loaded, then this method does nothing.

10.95 TGradient Component

Unit: WebUI

Inherits From TElementAttribute

The TGradient class represents a background gradient for a UI element or control. Background gradients can be linear or radial, and are comprised of 2 or more color stops.

Properties	Methods	Events
Angle	SetToDefault	
AutoCenter		
CenterX		
CenterY		
ColorStops		
GradientType		

TGradient.Angle Property

```
property Angle: Integer
```

Specifies the angle of the linear gradient in degrees. The angle is the direction in which the linear gradient will transition from the first color stop to the last color stop, with both 0 degrees and 360 degrees being the top of the background area of a UI element or control. The gradient is a linear gradient when the GradientType property is set to gtLinear.

TGradient.AutoCenter Property

```
property AutoCenter: Boolean
```

Specifies whether the radial gradient should be auto-centered in the exact middle of the background area of a UI element or control. The gradient is a radial gradient when the GradientType property is set to gtRadial.

TGradient.CenterX Property

```
property CenterX: Integer
```

Specifies the horizontal position of the center of the radial gradient, relative to the background area of a UI element or control. The gradient is a radial gradient when the GradientType property is set to gtRadial.

TGradient.CenterY Property

```
property CenterY: Integer
```

Specifies the vertical position of the center of the radial gradient, relative to the background area of a UI element or control. The gradient is a radial gradient when the GradientType property is set to gtRadial.

TGradient.ColorStops Property

```
property ColorStops: TGradientColorStops
```

Specifies the color stops for the gradient. Each color stop is represented by a color and a position (0-100), with each color transitioning to the next at the specified positions. Each position is relative to the existing width and height of the background area for a UI element or control, and adjusts proportionally as the background area is resized.

> **Note**
> A gradient always requires at least 2 color stops at positions 0 and 100.

TGradient.GradientType Property

```
property GradientType: TGradientType
```

Specifies the type of gradient, linear or radial.

TGradient.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the gradient's properties to their default values.

10.96 TGradientColorStop Component

Unit: WebUI

Inherits From TElementAttribute

The TGradientColorStop class represents a background gradient color stop for a UI element or control.

Properties	Methods	Events
Color	SetToDefault	
Position		

TGradientColorStop.Color Property

```
property Color: TColor
```

Specifies the color for the gradient color stop.

TGradientColorStop.Position Property

```
property Position: Integer
```

Specifies the relative position for the gradient color stop.

TGradientColorStop.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the gradient color stop's properties to their default values.

10.97 TGradientColorStops Component

Unit: WebUI

Inherits From TElementAttribute

The TGradientColorStops class represents the background gradient color stops for a UI element or control.

Properties	Methods	Events
ColorStop	Add	
Count	Remove	
	RemoveAll	
	SetToDefault	

TGradientColorStops.ColorStop Property

```
property ColorStop[Index: Integer]: TGradientColorStop
```

Accesses a gradient color stop by its index position (0 to Count property).

TGradientColorStops.Count Property

```
property Count: Integer
```

Indicates the number of defined gradient color stops.

TGradientColorStops.Add Method

```
function Add: TGradientColorStop
```

Use this method to add a new gradient color stop. The color stop will be initialized with a Color value of clBlack and a Position value of 0.

TGradientColorStops.Remove Method

```
procedure Remove(Index: Integer)
```

Use this method to remove an existing gradient color stop by its index position in the color stops.

TGradientColorStops.RemoveAll Method

```
procedure RemoveAll
```

Use this method to remove all existing gradient color stops.

TGradientColorStops.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the gradient color stops' properties to their default values.

10.98 TGrid Component

Unit: WebGrids

Inherits From TGridControl

The TGrid component represents a grid control. A grid can be un-bound and used to display and update a matrix of cells as strings, or can be bound to TDataSet component instances in order to display and update the rows in the dataset.

Each grid instance has its own defined set of columns and, if the grid is bound to a dataset, each column can be bound to a specific dataset column. Grid columns can also be used to sort a dataset by a specific set of columns via their SortDirection property.

Properties	Methods	Events
AllowAppends		OnAnimationComplete
AllowDeletes		OnAnimationsComplete
AllowInserts		OnClick
Background		OnColumnChanged
Border		OnDblClick
ColumnHeaders		OnHide
ColumnHeadersHeight		OnKeyDown
Corners		OnKeyPress
Cursor		OnKeyUp
DataSet		OnMouseDown
Enabled		OnMouseEnter
Hint		OnMouseLeave
MultiSelect		OnMouseMove
ReadOnly		OnMouseUp
RowHeight		OnMouseWheel
RowSelect		OnMove
ScrollBars		OnRowChanged
ScrollSupport		OnScroll
ShowLines		OnShow
TabOrder		OnSize
TabStop		OnTouchCancel
WantTabs		OnTouchEnd
		OnTouchMove
		OnTouchScroll
		OnTouchStart

TGrid.AllowAppends Property

```
property AllowAppends: Boolean
```

Specifies whether the grid should allow new rows to be appended by the user moving the active row index past the last row in the grid. The default value is True.

TGrid.AllowDeletes Property

```
property AllowDeletes: Boolean
```

Specifies whether the grid should allow rows to be deleted by the user using the Ctrl-Delete key combination in the grid. The default value is True.

TGrid.AllowInserts Property

```
property AllowInserts: Boolean
```

Specifies whether the grid should allow new rows to be inserted by the user using the Insert or Shift-Insert key combination in the grid. The default value is True.

TGrid.Background Property

```
property Background: TBackground
```

Specifies the background for the control.

TGrid.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TGrid.ColumnHeaders Property

```
property ColumnHeaders: Boolean
```

Specifies whether column headers should be shown for the columns in the grid.

TGrid.ColumnHeadersHeight Property

```
property ColumnHeadersHeight: Integer
```

Specifies the height of the column headers in the grid.

TGrid.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TGrid.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TGrid.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the grid to. The default value is nil.

Note
In order to actually show data from the dataset, the grid Columns must also have their DataColumn property set to valid dataset column names.

TGrid.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TGrid.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TGrid.MultiSelect Property

```
property MultiSelect: Boolean
```

Specifies whether multiple rows can be selected in the grid. The SelectedCount property can be examined to find out how many rows are selected and the Selected property can be examined to find out which rows are selected.

Note
You can only use the multi-select functionality when the RowSelect property is True. Setting the RowSelect property to False will automatically set the MultiSelect property to False also.

TGrid.ReadOnly Property

```
property ReadOnly: Boolean
```

Specifies whether the grid's rows and columns can be modified by the user. The default value is False.

Note
The grid rows and columns can always be programmatically modified.

TGrid.RowHeight Property

```
property RowHeight: Integer
```

Indicates the height, in pixels, of each visible row in the grid.

TGrid.RowSelect Property

```
property RowSelect: Boolean
```

Specifies that the active selection should always encompass the entire current row. The default value is False.

TGrid.ScrollBars Property

```
property ScrollBars: TScrollBars
```

Specifies which scrollbars to show, if any.

Note
Even if this property is set to sbHorizontal, sbVertical, or sbBoth, a scrollbar will only be shown if the size of the contents of the control exceed the client rectangle for the control.

TGrid.ScrollSupport Property

```
property ScrollSupport: TScrollSupport
```

Specifies the directions in which the control can be scrolled, if any.

> **Note**
> This property only applies to scrolling via touch and mouse wheel movements, and does not apply to programmatic scrolling or scrolling via the scroll bars themselves.

TGrid.ShowLines Property

```
property ShowLines: Boolean
```

Specifies that horizontal and vertical lines should be used to separate the various grid columns and rows.

TGrid.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TGrid.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TGrid.WantTabs Property

```
property WantTabs: Boolean
```

Specifies that the tab key can be used to navigate from cell to cell in the grid. The default value is True.

Note
The RowSelect property must be False for this property setting to take effect. If the RowSelect property is True, then the tab key will cause the input focus to move to the next control within the parent container control for the grid.

TGrid.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TGrid.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TGrid.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TGrid.OnColumnChanged Event

```
property OnColumnChanged: TNotifyEvent
```

This event is triggered whenever the ColumnIndex property changes.

TGrid.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TGrid.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TGrid.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when the control has input focus and the user presses a key or key combination.

TGrid.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TGrid.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when the control has input focus and the user releases a key or key combination.

TGrid.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TGrid.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TGrid.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TGrid.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TGrid.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TGrid.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

This event is triggered whenever the mouse wheel is rotated forward or backward.

TGrid.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TGrid.OnRowChanged Event

```
property OnRowChanged: TNotifyEvent
```

This event is triggered whenever the RowIndex property changes.

TGrid.OnScroll Event

```
property OnScroll: TNotifyEvent
```

This event is triggered whenever a scrollable control is scrolled horizontally or vertically.

TGrid.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TGrid.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TGrid.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TGrid.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TGrid.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TGrid.OnTouchScroll Event

```
property OnTouchScroll: TTouchScrollEvent
```

This event is triggered whenever a touch moves in any direction over a touch-scroll-enabled control.

TGrid.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.99 TGridCell Component

Unit: WebGrids

Inherits From TControl

The TGridCell component represents a visible grid cell. A grid cell is used to display the values for a given grid column in a grid control. The Background and Font properties can be modified in a TGridColumn OnCellUpdate event handler to affect how data is displayed in the grid.

Properties	Methods	Events
Background		
Data		
Font		
Index		

TGridCell.Background Property

```
property Background: TBackground
```

Specifies the background of the cell.

TGridCell.Data Property

```
property Data: String
```

Indicates the current contents of the cell.

TGridCell.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the cell.

TGridCell.Index Property

```
property Index: Integer
```

Indicates the index of the cell in the grid column.

10.100 TGridColumn Component

Unit: WebGrids

Inherits From TBindableColumnControl

The TGridColumn component represents a grid column. A grid column can be un-bound and used to display and update a vertical list of cells as strings, or can be bound to one of the TDataSet Columns in order to display and update the column values for rows in the dataset.

Grid columns can be used to sort a dataset by a specific set of columns via their SortDirection property.

The ControlType property controls what type of control, if any, will be used to edit the contents of the cells for the grid column.

Properties	Methods	Events
Alignment	ToggleSortDirection	OnButtonClick
AllowSize		OnCellUpdate
AutoDropDown		OnCompare
CalendarDefaultView		OnDropDownHide
CalendarHeight		OnDropDownShow
CalendarWidth		OnHeaderClick
ControlType		OnHide
DataColumn		OnShow
Direction		OnSize
DropDownItemCount		
DropDownPosition		
DropDownVisible		
Enabled		
Font		
Header		
ImageLayout		
Index		
Items		
ItemsSorted		
MaxLength		
ParentGrid		
ReadOnly		
SingleClickToggle		
SortDirection		
SortIndex		
SpellCheck		
StretchToFit		
ValueSelected		
ValueUnselected		
Wrap		

TGridColumn.Alignment Property

```
property Alignment: TContentAlignment
```

Specifies the alignment of the cell text for the grid column.

TGridColumn.AllowSize Property

```
property AllowSize: Boolean
```

Specifies whether the grid column can be sized by moving the mouse over the right border of the grid column header. The default value is True.

TGridColumn.AutoDropDown Property

```
property AutoDropDown: Boolean
```

Specifies that the drop-down list should automatically be shown when the user starts typing in the grid column's edit control. This property is only applicable when the ControlType property is set to ctEditComboBox.

TGridColumn.CalendarDefaultView Property

```
property CalendarDefaultView: TCalendarView
```

Specifies the default view for the drop-down calendar control when the ControlType property is set to ctDateEditCombobBox. The default view determines both the initial view shown in the calendar after it is created, as well as the minimum view that the user is permitted to navigate to. The default value is cvMonth.

TGridColumn.CalendarHeight Property

```
property CalendarHeight: Integer
```

Specifies the height of the drop-down calendar control when the ControlType property is set to ctDateEditCombobBox.

TGridColumn.CalendarWidth Property

```
property CalendarWidth: Integer
```

Specifies the width of the drop-down calendar control when the ControlType property is set to ctDateEditCombobBox.

TGridColumn.ControlType Property

```
property ControlType: TGridColumnControlType
```

Specifies the control type to use for any in-place editing for the column. The default value is ctNone, which means that the column cannot be edited.

If the ControlType is set to ctLink, then the cell or bound dataset values should be in the format of:

```
<URL> [; <Optional Title>]
```

If the ControlType is set to ctImage, then the cell or bound dataset values should be in the format of:

```
<URL> [; <Optional Hint>]
```

Note

If the <URL> specified in the cell or bound dataset value for an image column (ControlType=ctImage) contains a data-URL value (for example, data:image/png;base64,....), then you cannot specify an optional hint for the image.

TGridColumn.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TGridColumn.Direction Property

```
property Direction: TContentDirection
```

Specifies the direction in which the grid column's content is displayed/edited.

TGridColumn.DropDownItemCount Property

```
property DropDownItemCount: Integer
```

Specifies the number of visible items to display in the drop-down list of the grid column's edit control when the ControlType property is set to ctEditComboBox.

TGridColumn.DropDownPosition Property

```
property DropDownPosition: TDropDownPosition
```

Specifies the position of the drop-down list/calendar of the grid column's edit control when the ControlType property is set to ctEditComboBox or ctDateEditCombobBox.

TGridColumn.DropDownVisible Property

```
property DropDownVisible: Boolean
```

Indicates whether the drop-down list of the grid column's edit control is visible when the ControlType property is set to ctEditComboBox or ctDialogEditComboBox.

TGridColumn.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the column is enabled or disabled. When a grid column is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TGridColumn.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TGridColumn.Header Property

```
property Header: TGridHeader
```

Specifies the header properties for the grid column. The grid column header is only visible when the TGrid ColumnHeaders property is set to True.

TGridColumn.ImageLayout Property

```
property ImageLayout: TContentLayout
```

Specifies the layout properties of the image content contained within the column when the ControlType property is set to ctImage.

TGridColumn.Index Property

```
property Index: Integer
```

Indicates the index of the column in the list of grid columns.

TGridColumn.Items Property

```
property Items: TStrings
```

Specifies the list of items to use in the drop-down list of the grid column's edit control when the ControlType property is set to ctEditComboBox.

TGridColumn.ItemsSorted Property

```
property ItemsSorted: Boolean
```

Specifies whether the list of items to display in the drop-down list of the grid column's edit control are sorted when the ControlType property is set to ctEditComboBox.

TGridColumn.MaxLength Property

```
property MaxLength: Integer
```

Specifies the maximum allowable length for any text in the grid column's edit control when the ControlType property is set to ctEdit, ctEditComboBox, ctDialogEditComboBox, or ctMultiLineEdit.

TGridColumn.ParentGrid Property

```
property ParentGrid: TGridControl
```

Indicates the parent grid control that contains the column, or nil if the column has not been assigned to a grid control.

TGridColumn.ReadOnly Property

```
property ReadOnly: Boolean
```

Specifies whether or not the grid column can be modified.

TGridColumn.SingleClickToggle Property

```
property SingleClickToggle: Boolean
```

Specifies that whether a click (True) or double-click (False) is required in order to toggle the selected state of the grid column's control. The default value is False. This property is used when the ControlType property is set to ctCheckBox.

TGridColumn.SortDirection Property

```
property SortDirection: TSortDirection
```

Specifies the sort for the column. If the grid is data-bound (DataSet property is not nil) and the column has been assigned a valid column name in the DataColumn, then assigning sdAscending or sdDescending to this property will cause the dataset to add the column specified in the DataColumn property to the active sort for the dataset. Assigning sdNone to this property will cause the dataset to remove the column specified in the DataColumn property from the active sort for the dataset.

If the grid is un-bound, then assigning sdAscending or sdDescending to this property will cause the grid to add the column to the active sort. Assigning sdNone to this property will cause the grid to remove the column from the active sort.

The default value is sdNone.

If the ColumnHeaders property is True when this property is changed, then the column header will reflect the sort status of the column.

Note

The TDataSet Sort method is automatically called by the grid when the grid is data-bound and this property is changed. If the grid is un-bound, then the TGrid SortRows method is automatically called when this property is changed.

TGridColumn.SortIndex Property

```
property SortIndex: Integer
```

If the SortDirection property is not equal to sdNone, then this property indicates the position of the column in the active sort for an un-bound grid.

TGridColumn.SpellCheck Property

```
property SpellCheck: Boolean
```

Specifies whether spell-checking is enabled for the grid column's edit control when the ControlType property is set to ctEdit, ctEditComboBox, ctDialogEditComboBox, or ctMultiLineEdit.

TGridColumn.StretchToFit Property

```
property StretchToFit: Boolean
```

Specifies whether the column should automatically stretch to fit against the right edge of the client rectangle for the grid.

TGridColumn.ValueSelected Property

```
property ValueSelected: String
```

Specifies the textual value to use for the selected state when reading and writing data to and from the DataColumn that the control is bound to. This property is used when the ControlType property is set to ctCheckBox. The default value is 'True'.

TGridColumn.ValueUnselected Property

```
property ValueUnselected: String
```

Specifies the textual value to use for the unselected state when reading and writing data to and from the DataColumn that the control is bound to. This property is used when the ControlType property is set to ctCheckBox. The default value is 'False'.

TGridColumn.Wrap Property

```
property Wrap: Boolean
```

Specifies whether the text in the column should wrap if it exceeds the width of the column. The default value is False.

TGridColumn.ToggleSortDirection Method

```
procedure ToggleSortDirection(AClear: Boolean=False)
```

Use this method to toggle the column's SortDirection property.

TGridColumn.OnButtonClick Event

```
property OnButtonClick: TClickEvent
```

This event is triggered whenever the associated combo button is clicked for any grid column whose ControlType is ctEditComboBox, ctDateEditComboBox, or ctDialogEditComboBox.

TGridColumn.OnCellUpdate Event

```
property OnCellUpdate: TGridColumnCellEvent
```

This event is triggered whenever a cell in the grid column is updated, and can be used to override the default display of the grid cell parameter by modifying its background and/or font.

TGridColumn.OnCompare Event

```
property OnCompare: TGridColumnCompareEvent
```

This event is triggered whenever the grid column is sorted in an un-bound grid, and can be used to override the default sorting of the string column values.

TGridColumn.OnDropDownHide Event

```
property OnDropDownHide: TNotifyEvent
```

This event is triggered when the associated drop-down control is hidden.

TGridColumn.OnDropDownShow Event

```
property OnDropDownShow: TNotifyEvent
```

This event is triggered when the associated drop-down control is shown.

TGridColumn.OnHeaderClick Event

```
property OnHeaderClick: TGridHeaderClickEvent
```

This event is triggered when the grid column's header is clicked. Return False from any event handler attached to this event in order to prevent the default action when the header is clicked.

TGridColumn.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TGridColumn.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TGridColumn.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

10.101 TGridControl Component

Unit: WebGrids

Inherits From TBindableControl

The TGridControl control is the base class for grids, and contains all of the core grid functionality in the form of public methods and protected properties/events that descendant classes can use to create customized grids.

Properties	Methods	Events
AlwaysShowControls	AddColumnsFromDataSet	
ColumnControlActive	AppendRow	
ColumnControlVisible	DeleteRow	
ColumnCount	FirstColumn	
ColumnIndex	FirstRow	
Columns	GetGridColumn	
RowCount	GetRows	
RowIndex	HideColumnControl	
RowOffset	InsertRow	
Rows	LastColumn	
RowVisible	LastRow	
Selected	LoadRows	
SelectedCount	MakeColumnVisible	
SortCaseInsensitive	MakeRowVisible	
Sorted	NewColumn	
SortLocaleInsensitive	NextColumn	
VisibleColumnCount	NextPage	
VisibleColumns	NextRow	
	PriorColumn	
	PriorPage	
	PriorRow	
	RemoveColumn	
	ScrollNext	
	ScrollNextPage	
	ScrollPrior	

	ScrollPriorPage	
	ScrollTo	
	SelectAll	
	SelectRange	
	SetToColumn	
	SetToRow	
	ShowColumnControl	
	SortRows	
	ToggleSelected	

TGridControl.AlwaysShowControls Property

```
property AlwaysShowControls: Boolean
```

Specifies whether the grid should always show any column controls for the grid columns. A grid column has a control associated with it when the TGridColumn ControlType property is not equal to ctNone.

Note
Not all column controls offer text editing capabilities. Some are used strictly for displaying the data in the grid column, while others like the checkbox control offer editing via double-click interactions.

TGridControl.ColumnControlActive Property

```
property ColumnControlActive: Boolean
```

Indicates whether any grid column control is currently active. The control for a grid column is controlled via the TGridColumn ControlType property, and the grid column controls can be made visible by calling the ShowColumnControl method. This property can only be true if the ColumnControlVisible property is True.

Note
Even if the ColumnControlVisible property is True, it is possible that there won't be any active column controls because row selection is turned on for the grid control or the column doesn't use a control for display or editing. Use this property to determine if any grid column controls are actually active.

TGridControl.ColumnControlVisible Property

```
property ColumnControlVisible: Boolean
```

Indicates whether any grid column controls should be visible. The control for a grid column is controlled via the TGridColumn ControlType property, and the grid column controls can be made visible by calling the ShowColumnControl method.

Note
Even if the ColumnControlVisible property is True, it is possible that there won't be any active column controls because row selection is turned on for the grid control or the column doesn't use a control for display or editing. You can use the ColumnControlActive property to determine if any grid column controls are actually active.

TGridControl.ColumnCount Property

```
property ColumnCount: Integer
```

Indicates the number of columns in the grid control.

TGridControl.ColumnIndex Property

```
property ColumnIndex: Integer
```

Specifies the current column index for the grid control. If there is no current column, this property will be -1.

TGridControl.Columns Property

```
property Columns[AIndex: Integer]: TGridColumn

property Columns[const AName: String]: TGridColumn
```

Contains the defined columns for the grid control.

TGridControl.RowCount Property

```
property RowCount: Integer
```

Indicates the number of rows in the grid control. If the grid control has been bound to a TDataSet instance, then this property will be equal to the RowCount property of the dataset.

TGridControl.RowIndex Property

```
property RowIndex: Integer
```

Specifies the current row index for the grid control. If there is no current row, this property will be -1.

TGridControl.RowOffset Property

```
property RowOffset: Integer
```

Specifies the current row offset for the grid control. The row offset is the the index of the row that is the first visible row. If there are now rows present in the grid control, this property will be 0.

TGridControl.Rows Property

```
property Rows: TGridRows
```

Contains the rows for the grid when the grid control is un-bound.

TGridControl.RowVisible Property

```
property RowVisible: Boolean
```

Indicates whether the current row, specified by the RowIndex property, is visible.

TGridControl.Selected Property

```
property Selected[ARowIndex: Integer]: Boolean
```

Accesses the selection state of each row in the grid by its index.

TGridControl.SelectedCount Property

```
property SelectedCount: Integer
```

Indicates the number of selected rows in the grid.

TGridControl.SortCaseInsensitive Property

```
property SortCaseInsensitive: Boolean
```

Specifies whether or not an active sort on an un-bound grid, as indicated by the Sorted property, should be case-sensitive.

Note
Changing this property will trigger a sort on an un-bound grid if the Sorted property is True.

TGridControl.Sorted Property

```
property Sorted: Boolean
```

Indicates whether any of the Columns in the grid control have their SortDirection property set to sdAscending or sdDescending.

Note
This property does **not** indicate whether an un-bound grid control has actually been sorted yet via the Sort method. The sorting is designed this way in order to allow multiple columns to be designated as sort columns without triggering an automatic sort operation each time.

TGridControl.SortLocaleInsensitive Property

```
property SortLocaleInsensitive: Boolean
```

Specifies whether or not an active sort on an un-bound grid, as indicated by the Sorted property, should be locale-sensitive.

Note
Changing this property will trigger a sort on an un-bound grid if the Sorted property is True.

TGridControl.VisibleColumnCount Property

```
property VisibleColumnCount: Integer
```

Indicates the number of visible columns in the grid control.

TGridControl.VisibleColumns Property

```
property VisibleColumns[AIndex: Integer]: TGridColumn
```

Accesses the visible columns in the grid control by their index into the list of visible columns.

TGridControl.AddColumnsFromDataSet Method

```
procedure AddColumnsFromDataSet
```

Use this method to quickly create grid columns for all defined columns in the dataset referenced in the DataSet property.

Note

This method does **not** clear out any existing grid columns before creating the new columns.

TGridControl.AppendRow Method

```
procedure AppendRow
```

Use this method to append a row to the end of any existing rows in the grid control.

TGridControl.DeleteRow Method

```
procedure DeleteRow
```

Use this method to delete the current row, represented by the RowIndex property, from the grid control.

TGridControl.FirstColumn Method

```
function FirstColumn: Boolean
```

Use this method to change the current grid column to the first visible column in the grid, if one exists, and update the ColumnIndex property accordingly.

TGridControl.FirstRow Method

```
procedure FirstRow(ShiftKey, CtrlKey: Boolean=False)
```

Use this method to change the current grid row to the first row in the grid, if one exists, and update the RowIndex property accordingly.

Note

The ShiftKey and CtrlKey parameters control how multiple row selection occurs in relation to the row navigation. When the ShiftKey parameter is True, then the range of selected rows is extended to include all rows between the current row index and the row at the new index. When the CtrlKey parameter is True, then the set of selected rows is extended to include the new row at the new index.

TGridControl.GetGridColumn Method

```
function GetGridColumn(Value: TDataColumn): TGridColumn
```

Use this method to lookup a grid column by which TDataSet column it is bound to.

TGridControl.GetRows Method

```
function GetRows: String
```

Use this method to retrieve the rows of an un-bound grid as a JSON string. The JSON is formatted as follows:

```
{   "rows": [
{ "GridColumn1": "Test 1", "GridColumn2": "100", "GridColumn3": "" },
{ "GridColumn1": "Test 2", "GridColumn2": "200", "GridColumn3": "" },
{ "GridColumn1": "Test 3", "GridColumn2": "300", "GridColumn3": "" }
] }
```

TGridControl.HideColumnControl Method

```
procedure HideColumnControl
```

Use this method to hide any column controls for the columns in the grid. The control for a grid column is controlled via the TGridColumn ControlType property.

TGridControl.InsertRow Method

```
procedure InsertRow
```

Use this method to insert a row at the existing RowIndex position in any existing rows in the grid control.

TGridControl.LastColumn Method

```
function LastColumn: Boolean
```

Use this method to change the current grid column to the last visible column in the grid, if one exists, and update the ColumnIndex property accordingly.

TGridControl.LastRow Method

```
procedure LastRow(ShiftKey, CtrlKey: Boolean=False)
```

Use this method to change the current grid row to the last row in the grid, if one exists, and update the RowIndex property accordingly.

Note
The ShiftKey and CtrlKey parameters control how multiple row selection occurs in relation to the row navigation. When the ShiftKey parameter is True, then the range of selected rows is extended to include all rows between the current row index and the row at the new index. When the CtrlKey parameter is True, then the set of selected rows is extended to include the new row at the new index.

TGridControl.LoadRows Method

```
procedure LoadRows(const RowData: String; Append: Boolean=False)
```

Use this method to load the rows of an un-bound grid from a JSON string. The JSON should be formatted as follows:

```
{   "rows": [
{ "GridColumn1": "Test 1", "GridColumn2": "100", "GridColumn3": "" },
{ "GridColumn1": "Test 2", "GridColumn2": "200", "GridColumn3": "" },
{ "GridColumn1": "Test 3", "GridColumn2": "300", "GridColumn3": "" }
] }
```

TGridControl.MakeColumnVisible Method

```
procedure MakeColumnVisible(AColumn: TGridColumn)
```

Use this method to ensure that the specified grid column is visible within the grid control.

TGridControl.MakeRowVisible Method

```
procedure MakeRowVisible
```

Use this method to ensure that the specified grid row is visible within the grid control.

TGridControl.NewColumn Method

```
function NewColumn: TGridColumn
```

Use this method to create a new column for the grid control and return it as the result. The new grid column will be appended to the existing list of columns defined in the Columns property.

TGridControl.NextColumn Method

```
function NextColumn(AWrap: Boolean=False): Boolean
```

Use this method to change the current grid column to the next visible column in the grid, if one exists, and update the ColumnIndex property accordingly. The AWrap parameter determines whether to move the current grid row to the next available row, if one exists, if there are no more visible columns in the grid.

TGridControl.NextPage Method

```
procedure NextPage(ShiftKey, CtrlKey: Boolean=False)
```

Use this method to change the current grid row to the start of the next page of rows in the grid, if one exists, and update the RowIndex property accordingly.

> **Note**
> The ShiftKey and CtrlKey parameters control how multiple row selection occurs in relation to the row navigation. When the ShiftKey parameter is True, then the range of selected rows is extended to include all rows between the current row index and the row at the new index. When the CtrlKey parameter is True, then the set of selected rows is extended to include the new row at the new index.

TGridControl.NextRow Method

```
procedure NextRow(ShiftKey, CtrlKey: Boolean=False; CanAppend:
      Boolean=False)
```

Use this method to change the current grid row to the next row in the grid, if one exists, and update the RowIndex property accordingly. The CanAppend parameter determines whether the grid should automatically call the AppendRow method if the current row index is equal to the last row index for the grid.

Note

The ShiftKey and CtrlKey parameters control how multiple row selection occurs in relation to the row navigation. When the ShiftKey parameter is True, then the range of selected rows is extended to include all rows between the current row index and the row at the new index. When the CtrlKey parameter is True, then the set of selected rows is extended to include the new row at the new index.

TGridControl.PriorColumn Method

```
function PriorColumn(AWrap: Boolean=False): Boolean
```

Use this method to change the current grid column to the prior visible column in the grid, if one exists, and update the ColumnIndex property accordingly. The AWrap parameter determines whether to move the current grid row to the prior available row, if one exists, if there are no more visible columns in the grid.

TGridControl.PriorPage Method

```
procedure PriorPage(ShiftKey, CtrlKey: Boolean=False)
```

Use this method to change the current grid row to the end of the next page of rows in the grid, if one exists, and update the RowIndex property accordingly.

Note

The ShiftKey and CtrlKey parameters control how multiple row selection occurs in relation to the row navigation. When the ShiftKey parameter is True, then the range of selected rows is extended to include all rows between the current row index and the row at the new index. When the CtrlKey parameter is True, then the set of selected rows is extended to include the new row at the new index.

TGridControl.PriorRow Method

```
procedure PriorRow(ShiftKey, CtrlKey: Boolean=False)
```

Use this method to change the current grid row to the prior row in the grid, if one exists, and update the RowIndex property accordingly.

Note
The ShiftKey and CtrlKey parameters control how multiple row selection occurs in relation to the row navigation. When the ShiftKey parameter is True, then the range of selected rows is extended to include all rows between the current row index and the row at the new index. When the CtrlKey parameter is True, then the set of selected rows is extended to include the new row at the new index.

TGridControl.RemoveColumn Method

```
procedure RemoveColumn(AColumn: TGridColumn)
```

Use this method to remove the specified column from the grid control.

TGridControl.ScrollNext Method

```
procedure ScrollNext
```

Use this method to scroll the grid down by the height of one row.

TGridControl.ScrollNextPage Method

```
procedure ScrollNextPage
```

Use this method to scroll the grid down by the height of all visible rows in the grid.

TGridControl.ScrollPrior Method

```
procedure ScrollPrior
```

Use this method to scroll the grid up by the height of one row.

TGridControl.ScrollPriorPage Method

```
procedure ScrollPriorPage
```

Use this method to scroll the grid up by the height of all visible rows in the grid.

TGridControl.ScrollTo Method

```
procedure ScrollTo(ARowIndex: Integer)
```

Use this method to scroll the grid to the specified row in the grid control.

TGridControl.SelectAll Method

```
procedure SelectAll
```

Use this method to select all of the rows in the grid control.

TGridControl.SelectRange Method

```
procedure SelectRange(AFromRowIndex, AToRowIndex: Integer;
      AClear: Boolean=False)
```

Use this method to select the specified range of the rows in the grid control. The AClear parameter determines whether the existing set of selected rows should be cleared before the new range of rows is selected.

TGridControl.SetToColumn Method

```
procedure SetToColumn(AIndex: Integer)
```

Use this method to change the current grid column to the visible column at the specified index in the grid, if one exists, and update the ColumnIndex property accordingly.

TGridControl.SetToRow Method

```
procedure SetToRow(AIndex: Integer; ShiftKey, CtrlKey:
      Boolean=False)
```

Use this method to change the current grid row to the row at the specified index in the grid, if one exists, and update the RowIndex property accordingly.

Note
The ShiftKey and CtrlKey parameters control how multiple row selection occurs in relation to the row navigation. When the ShiftKey parameter is True, then the range of selected rows is extended to include all rows between the current row index and the row at the new index. When the CtrlKey parameter is True, then the set of selected rows is extended to include the new row at the new index.

TGridControl.ShowColumnControl Method

```
procedure ShowColumnControl
```

Use this method to show any column controls for the columns in the grid. The control for a grid column is controlled via the TGridColumn ControlType property. This method will change the ColumnControlVisible property to True.

Note
Even if the ColumnControlVisible property is True, it is possible that there won't be any active column controls because row selection is turned on for the grid control or the column doesn't use a control for display or editing. You can use the ColumnControlActive property to determine if any grid column controls are actually active.

TGridControl.SortRows Method

```
procedure SortRows
```

Use this method to sort an un-bound grid when the Sorted property is True and sort columns have been specified for the grid. The TGridColumn SortDirection property setting controls which columns are sorted, and how.

If no columns have a SortDirection property other than sdNone, then this method does nothing.

The SortCaseInsensitive and SortLocaleInsensitive properties control how the sort is performed.

Note
The SortRows method only needs to be called once after the sort directions are initially assigned for the grid columns. After that point, any row operations will automatically maintain the active sort.

TGridControl.ToggleSelected Method

```
procedure ToggleSelected(ARowIndex: Integer)
```

Use this method to toggle the selection state of the row at the specified index.

10.102 TGridHeader Component

Unit: WebGrids

Inherits From TControl

The TGridHeader component represents a grid column header. Grid column headers are only visible when the TGrid ColumnHeaders property is True.

Properties	Methods	Events
Alignment		
AllowClick		
Caption		
Font		
Hint		
Wrap		

TGridHeader.Alignment Property

```
property Alignment: TContentAlignment
```

Specifies the alignment of the caption for the grid column header.

TGridHeader.AllowClick Property

```
property AllowClick: Boolean
```

Specifies whether the grid column header is clickable. If the header is clickable, then each click will call the ToggleSortDirection method to change the sort direction to the next valid state.

TGridHeader.Caption Property

```
property Caption: String
```

Specifies the caption for the grid column header.

TGridHeader.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TGridHeader.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TGridHeader.Wrap Property

```
property Wrap: Boolean
```

Specifies whether the caption should wrap if it exceeds the width of the column header. The default value is False.

10.103 TGridRow Component

Unit: WebGrids

Inherits From TObject

The TGridRow component represents a grid row in an un-bound grid control, and is accessible via the TGridControl Rows property.

Properties	Methods	Events
Count	Create	
ID	GetJSON	
Value		

TGridRow.Count Property

```
property Count: Integer
```

Indicates the number of column values in the grid row.

TGridRow.ID Property

```
property ID: Integer
```

Specifies an ID that unique identifies the grid row.

TGridRow.Value Property

```
property Value[AIndex: Integer]: String
```

Accesses a specific column value by its column index in the grid row.

TGridRow.Create Method

```
constructor Create(ARows: TGridRows; AID: Integer)
```

Use this method to create a new instance of the TGridRow class. The ARows parameter indicates the parent grid rows instance that will manage the row, and the AID parameter indicates the unique ID used to identify the grid row.

TGridRow.GetJSON Method

```
function GetJSON: String
```

Use this method to retrieve the column values of the row as a JSON string. The JSON is formatted as follows:

```
{ "GridColumn1": "Test 1", "GridColumn2": "100", "GridColumn3": "" }
```

10.104 TGridRows Component

Unit: WebGrids

Inherits From TObject

The TGridRows component represents the grid rows in an un-bound grid control, and are accessible via the TGridControl Rows property.

Properties	Methods	Events
Count	BeginUpdate	
Row	Create	
	EndUpdate	

TGridRows.Count Property

```
property Count: Integer
```

Indicates the number of rows in an un-bound grid control.

TGridRows.Row Property

```
property Row[AIndex: Integer]: TGridRow
```

Accesses a grid row by by the specified index.

TGridRows.BeginUpdate Method

```
procedure BeginUpdate
```

Use this method to begin a batch update to the un-bound grid rows. Batch updates are useful in situations where many changes need to be made to the rows, and triggering grid updates on every change would result in performance issues. This method is reference-counted and every time it is called, an internal counter is incremented. Every time the EndUpdate method is called, the counter is decremented. Once the counter reaches zero, the grid will be notified that it needs to update its contents.

TGridRows.Create Method

```
constructor Create(AGrid: TGridControl)
```

Use this method to create a new instance of the TGridRows class. The AGrid parameter indicates the parent grid control instance that will manage the rows.

TGridRows.EndUpdate Method

```
procedure EndUpdate
```

Use this method to end a batch update to the un-bound grid rows. Batch updates are useful in situations where many changes need to be made to the rows, and triggering grid updates on every change would result in performance issues. This method is reference-counted and every time it is called, an internal counter is decremented. Every time the BeginUpdate method is called, the counter is incremented. Once the counter reaches zero, the grid will be notified that it needs to update its contents.

10.105 TGroupPanel Component

Unit: WebCtnrs

Inherits From TGroupPanelControl

The TGroupPanel component represents a group panel control for organizing sets of controls like radio buttons into a container with a caption.

Properties	Methods	Events
ActivateOnClick		OnAnimationComplete
Background		OnAnimationsComplete
Caption		OnClick
Cursor		OnDblClick
Font		OnHide
Hint		OnKeyDown
InsetShadow		OnKeyPress
Opacity		OnKeyUp
OutsetShadow		OnMouseDown
Padding		OnMouseEnter
TabOrder		OnMouseLeave
TabStop		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TGroupPanel.ActivateOnClick Property

```
property ActivateOnClick: Boolean
```

Specifies whether the control should automatically be brought to the front when it, or any child controls, are clicked.

TGroupPanel.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TGroupPanel.Caption Property

```
property Caption: String
```

Specifies the caption for the group panel control.

TGroupPanel.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TGroupPanel.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TGroupPanel.Hint Property

```
property Hint: String
```

TGroupPanel.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Specifies the inset shadow for the control.

TGroupPanel.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TGroupPanel.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the control.

TGroupPanel.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

TGroupPanel.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TGroupPanel.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TGroupPanel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TGroupPanel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TGroupPanel.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TGroupPanel.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TGroupPanel.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TGroupPanel.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when a child control has input focus and the user presses a key or key combination.

TGroupPanel.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when a child control has input focus and presses/releases a key or key combination.

TGroupPanel.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when a child control has input focus and the user releases a key or key combination.

TGroupPanel.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TGroupPanel.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TGroupPanel.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TGroupPanel.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TGroupPanel.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TGroupPanel.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TGroupPanel.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TGroupPanel.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TGroupPanel.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TGroupPanel.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TGroupPanel.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TGroupPanel.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.106 TGroupPanelControl Component

Unit: WebCtnrs

Inherits From TControl

The TGroupPanelControl control is the base class for group panels, and contains all of the core group panel functionality in the form of public methods and protected properties/events that descendant classes can use to create customized group panels.

Properties	Methods	Events

10.107 THeaderPanel Component

Unit: WebCtnrs

Inherits From THeaderPanelControl

The THeaderPanel component represents a header panel control that acts as a container for other controls.

Properties	Methods	Events
Background		OnAnimationComplete
Border		OnAnimationsComplete
Corners		OnClick
Cursor		OnDblClick
Hint		OnHide
InsetShadow		OnKeyDown
Opacity		OnKeyPress
OutsetShadow		OnKeyUp
Padding		OnMouseDown
TabOrder		OnMouseEnter
TabStop		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

THeaderPanel.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

THeaderPanel.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

THeaderPanel.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

THeaderPanel.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

THeaderPanel.Hint Property

```
property Hint: String
```

THeaderPanel.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Specifies the inset shadow for the control.

THeaderPanel.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

THeaderPanel.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the control.

THeaderPanel.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

THeaderPanel.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

THeaderPanel.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

THeaderPanel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

THeaderPanel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

THeaderPanel.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

THeaderPanel.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

THeaderPanel.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

THeaderPanel.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when a child control has input focus and the user presses a key or key combination.

THeaderPanel.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when a child control has input focus and presses/releases a key or key combination.

THeaderPanel.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when a child control has input focus and the user releases a key or key combination.

THeaderPanel.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

THeaderPanel.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

THeaderPanel.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

THeaderPanel.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

THeaderPanel.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

THeaderPanel.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

THeaderPanel.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

THeaderPanel.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

THeaderPanel.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

THeaderPanel.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

THeaderPanel.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

THeaderPanel.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.108 THeaderPanelControl Component

Unit: WebCtnrs

Inherits From TControl

The THeaderPanelControl control is the base class for header panels, and contains all of the core header panel functionality in the form of public methods and protected properties/events that descendant classes can use to create customized header panels.

Properties	Methods	Events

10.109 THiddenInputElement Component

Unit: WebUI

Inherits From TInputElement

The THiddenInputElement class is the element class for hidden HTML form input elements. Hidden form input elements are useful for programmatically adding HTML form values, and the Elevate Web Builder Component library uses them for custom controls like the virtual TListBox control that have no counterpart in standard HTML form input elements.

> **Note**
> This element does not provide support for hidden HTML form inputs at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events

10.110 THTMLForm Component

Unit: WebBrwsr

Inherits From THTMLFormControl

The THTMLForm component represents an HTML form control. HTML forms are containers for any input controls whose input values can be submitted as HTML form values to the web server.

Properties	Methods	Events
Background		OnAnimationComplete
Border		OnAnimationsComplete
Corners		OnClick
Cursor		OnDblClick
Encoding		OnHide
InsetShadow		OnMouseDown
Method		OnMouseEnter
Opacity		OnMouseLeave
Output		OnMouseMove
Padding		OnMouseUp
TabOrder		OnMove
TabStop		OnShow
URL		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

THTMLForm.Background Property

```
property Background: TBackground
```

Specifies the background for the control.

THTMLForm.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

THTMLForm.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

THTMLForm.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

THTMLForm.Encoding Property

```
property Encoding: THTMLFormEncoding
```

Specifies the MIME encoding type of the form data that is sent when the Submit method is called.

THTMLForm.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Specifies the inset shadow for the control.

THTMLForm.Method Property

```
property Method: THTMLFormMethod
```

Specifies the HTTP method used for the form submittal when the Submit method is called.

THTMLForm.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

THTMLForm.Output Property

```
property Output: TBrowser
```

Specifies a TBrowser instance that should accept all output from the form submittal request to the web server when the Submit method is called. The default value is nil.

THTMLForm.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

THTMLForm.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

THTMLForm.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

THTMLForm.URL Property

```
property URL: String
```

Specifies the URL to use for the form submittal when the Submit method is called. The default value is ''.

THTMLForm.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

THTMLForm.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

THTMLForm.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

THTMLForm.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

THTMLForm.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

THTMLForm.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

THTMLForm.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

THTMLForm.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

THTMLForm.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

THTMLForm.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

THTMLForm.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

THTMLForm.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

THTMLForm.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

THTMLForm.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

THTMLForm.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

THTMLForm.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

THTMLForm.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.111 THTMLFormControl Component

Unit: WebBrwsr

Inherits From TControl

The THTMLFormControl control is the base class for HTML forms, and contains all of the core HTML form functionality in the form of public methods and protected properties/events that descendant classes can use to create customized HTML forms.

Properties	Methods	Events
	Reset	
	Submit	

THTMLFormControl.Reset Method

```
procedure Reset
```

Use this method to reset the HTML form values associated with any input controls contained within the HTML form control. Resetting the form values will set them all to an empty string ('').

THTMLFormControl.Submit Method

```
procedure Submit
```

Use this method to submit the HTML form values associated with any input controls contained within the HTML form control.

10.112 THTMLLabel Component

Unit: WebLabels

Inherits From THTMLLabelControl

The THTMLLabel component represents an HTML label control. An HTML label control displays arbitrary HTML content using a specific font and formatting, including alignment and wrapping.

Properties	Methods	Events
AutoSize		OnAnimationComplete
Background		OnAnimationsComplete
Border		OnClick
Content		OnHide
Corners		OnMouseDown
Cursor		OnMouseEnter
DataColumn		OnMouseLeave
DataSet		OnMouseMove
Font		OnMouseUp
Format		OnMove
Hint		OnShow
Opacity		OnSize
OutsetShadow		OnTouchCancel
Padding		OnTouchEnd
		OnTouchMove
		OnTouchStart

THTMLLabel.AutoSize Property

```
property AutoSize: TAutoSize
```

Specifies how (if at all) the control should automatically be sized based upon the Content and Format properties.

THTMLLabel.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

THTMLLabel.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

THTMLLabel.Content Property

```
property Content: TContent
```

Specifies the HTML content to display in the control. The default value is ''.

THTMLLabel.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

THTMLLabel.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

THTMLLabel.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

THTMLLabel.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

THTMLLabel.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

THTMLLabel.Format Property

```
property Format: TFormat
```

Specifies the content formatting to use for the control's Content.

THTMLLabel.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

THTMLLabel.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

THTMLLabel.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the control.

THTMLLabel.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

THTMLLabel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

THTMLLabel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

THTMLLabel.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

THTMLLabel.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

THTMLLabel.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

THTMLLabel.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

THTMLLabel.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

THTMLLabel.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

THTMLLabel.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

THTMLLabel.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

THTMLLabel.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

THTMLLabel.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

THTMLLabel.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

THTMLLabel.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

THTMLLabel.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

THTMLLabel.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.113 THTMLLabelControl Component

Unit: WebLabels

Inherits From TBindableColumnControl

The THTMLLabelControl control is the base class for HTML label controls, and contains all of the core HTML label functionality in the form of public methods and protected properties/events that descendant classes can use to create customized HTML label controls.

Properties	Methods	Events

10.114 TIcon Component

Unit: WebIcons

Inherits From TIconControl

The TIcon component represents an icon control. An icon control displays an icon, referenced by the Icon property, that contains a single background image or a single font icon. Icons are defined and stored in the Elevate Web Builder icon library.

Properties	Methods	Events
Cursor		OnAnimationComplete
Hint		OnAnimationsComplete
Icon		OnClick
Opacity		OnDblClick
		OnHide
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TIcon.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TIcon.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TIcon.Icon Property

```
property Icon: TIconProperties
```

Specifies the properties of the icon used with the control.

TIcon.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TIcon.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TIcon.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TIcon.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TIcon.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TIcon.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TIcon.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TIcon.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TIcon.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TIcon.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TIcon.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TIcon.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TIcon.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TIcon.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TIcon.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TIcon.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TIcon.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TIcon.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.115 TIconAnimation Component

Unit: WebCtrls

Inherits From TComponent

The TIconAnimation class represents the animation attributes to use for animating a specific property of icons. This class is used with various controls to specify the rotation animation properties of the icon used with the control.

Properties	Methods	Events
Duration		
Style		

TIconAnimation.Duration Property

```
property Duration: Integer
```

Specifies how long, in milliseconds, the animation should take to execute.

TIconAnimation.Style Property

```
property Style: TAnimationStyle
```

Specifies the style of the animation, which controls how the animation transforms a given UI element/control property. Currently, the supported styles include all of the standard easing transformations (including linear).

10.116 TIconButton Component

Unit: WebBtns

Inherits From TRepeatControl

The TIconButton component represents an icon button control. An icon button control displays an icon, referenced by the Icon property, as a button that can be pressed. Icons are stored in the Elevate Web Builder icon library.

Properties	Methods	Events
Cursor		OnAnimationComplete
Enabled		OnAnimationsComplete
Hint		OnClick
Icon		OnHide
RepeatClick		OnKeyDown
RepeatClickInterval		OnKeyUp
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TIconButton.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TIconButton.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the control is enabled or disabled. When a control is disabled, it is displayed in a disabled state. The default value is True.

TIconButton.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TIconButton.Icon Property

```
property Icon: TIconProperties
```

Specifies the properties of the icon used with the control.

TIconButton.RepeatClick Property

```
property RepeatClick: Boolean
```

Specifies whether the OnClick event handler should be triggered every RepeatClickInterval milliseconds while the button is pressed.

TIconButton.RepeatClickInterval Property

```
property RepeatClickInterval: Integer
```

Specifies the interval, in milliseconds, to trigger the OnClick event handler when the RepeatClick is True and the button is pressed.

TIconButton.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TIconButton.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TIconButton.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TIconButton.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TIconButton.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

TIconButton.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

TIconButton.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TIconButton.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TIconButton.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TIconButton.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TIconButton.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TIconButton.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TIconButton.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TIconButton.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TIconButton.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TIconButton.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TIconButton.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TIconButton.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.117 TIconControl Component

Unit: WebIcons

Inherits From TControl

The TIconControl control is the base class for icon controls, and contains all of the core icon control functionality in the form of public methods and protected properties/events that descendant classes can use to create customized icon controls.

Properties	Methods	Events

10.118 TIconLibrary Component

Unit: WebUI

Inherits From TComponent

The TIconLibrary class represents an icon library. An icon library is a list of embedded icon images stored as background images in control interface states. Each icon has a unique name and can be applied to a UI element so that the element displays the icon as its background image.

Properties	Methods	Events
	ApplyIcon	
	GetIconNames	

TIconLibrary.ApplyIcon Method

```
function ApplyIcon(AElement: TElement; const AIconName: String):
      Boolean
```

Use this method to apply an icon as the background image of an element. The AElement parameter specifies the element on which to apply the icon, and the AIconName parameter specifies the name of the icon to apply.

TIconLibrary.GetIconNames Method

```
function GetIconNames: array of String
```

Use this method to get a list of icon names available in the icon library as an array of strings.

10.119 TIconProperties Component

Unit: WebCtrls

Inherits From TComponent

The TIconProperties class represents the icon properties of icons that are embedded in various controls.

Properties	Methods	Events
Animating	StartAnimation	
Animation	StopAnimation	
FontIcon		
Height		
IconName		
Width		

TIconProperties.Animating Property

```
property Animating: Boolean
```

Indicates whether the icon's rotation animation is executing.

TIconProperties.Animation Property

```
property Animation: TIconAnimation
```

Specifies the rotation animation properties for the icon.

TIconProperties.FontIcon Property

```
property FontIcon: TFontIcon
```

Specifies the properties of the font icon (if a font icon is being used with the icon). These properties default to the pre-defined font icon properties of the icon specified by the IconName property, and may change when the IconName property is changed.

TIconProperties.Height Property

```
property Height: Integer
```

Specifies the height of the icon. This value defaults to the pre-defined height of the icon specified by the IconName property, and may change when the IconName property is changed.

TIconProperties.IconName Property

```
property IconName: String
```

Specifies the name of the icon to use with the control.

TIconProperties.Width Property

```
property Width: Integer
```

Specifies the width of the icon. This value defaults to the pre-defined width of the icon specified by the IconName property, and may change when the IconName property is changed.

TIconProperties.StartAnimation Method

```
procedure StartAnimation
```

Starts the rotation animation for the icon.

TIconProperties.StopAnimation Method

```
procedure StopAnimation
```

Stops the rotation animation for the icon.

10.120 TImage Component

Unit: WebBrwsr

Inherits From TWebControl

The TImage component represents an image control. An image control displays an image specified by the resource URL property at run-time.

Note
This control does not provide support for displaying images at design-time. If you wish to do so, you should specify the background image for a control. Only background images can be embedded in an Elevate Web Builder application.

Properties	Methods	Events
ActualHeight		OnAnimationComplete
ActualWidth		OnAnimationsComplete
Background		OnClick
Border		OnDblClick
ContentLayout		OnError
Corners		OnHide
Cursor		OnLoad
DataColumn		OnMouseDown
DataSet		OnMouseEnter
Hint		OnMouseLeave
InsetShadow		OnMouseMove
Loaded		OnMouseUp
Opacity		OnMove
OutsetShadow		OnShow
Padding		OnSize
URL		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart
		OnUnload

TImage.ActualHeight Property

```
property ActualHeight: Integer
```

Indicates the actual height of the loaded image when the URL property is specified and the Loaded property is True.

An event handler can be attached to the OnLoad event to execute code when the image is loaded.

TImage.ActualWidth Property

```
property ActualWidth: Integer
```

Indicates the actual width of the loaded image when the URL property is specified and the Loaded property is True.

An event handler can be attached to the OnLoad event to execute code when the image is loaded.

TImage.Background Property

```
property Background: TBackground
```

Specifies the background for the control.

TImage.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TImage.ContentLayout Property

```
property ContentLayout: TContentLayout
```

Specifies the layout properties of the image content contained within the control.

TImage.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TImage.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TImage.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TImage.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TImage.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TImage.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Specifies the inset shadow for the control.

TImage.Loaded Property

```
property Loaded: Boolean
```

Indicates whether the image specified by the URL property has been loaded.

An event handler can be attached to the OnLoad event to execute code when the image is loaded.

TImage.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TImage.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the control.

TImage.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

TImage.URL Property

```
property URL: String
```

Specifies the URL for the image. Whenever the URL property changes, the OnUnload event is triggered immediately. The OnLoad event is triggered once the image has been loaded.

TImage.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TImage.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TImage.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TImage.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TImage.OnError Event

```
property OnError: TNotifyEvent
```

This event is triggered when the download of the image encounters an error.

TImage.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TImage.OnLoad Event

```
property OnLoad: TNotifyEvent
```

This event is triggered when the image specified by the URL property has been completely loaded.

TImage.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TImage.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TImage.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TImage.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TImage.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TImage.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TImage.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TImage.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TImage.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TImage.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TImage.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TImage.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

TImage.OnUnload Event

```
property OnUnload: TNotifyEvent
```

This event is triggered when the currently-loaded image specified by the URL property has been unloaded.

10.121 TImageElement Component

Unit: WebUI

Inherits From TWebElement

The TImageElement class is the element class for image elements, and contains all of the image functionality in the form of public methods and properties/events that control classes can use to create image controls.

> **Note**
> This element does not provide support for images at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events
ActualHeight		
ActualWidth		

TImageElement.ActualHeight Property

```
property ActualHeight: Integer
```

Indicates the actual height of the loaded image when the URL property is specified and the Loaded property is True.

An event handler can be attached to the OnLoad event to execute code when the image is loaded.

TImageElement.ActualWidth Property

```
property ActualWidth: Integer
```

Indicates the actual width of the loaded image when the URL property is specified and the Loaded property is True.

An event handler can be attached to the OnLoad event to execute code when the image is loaded.

10.122 TInputControl Component

Unit: WebCtrls

Inherits From TBindableColumnControl

The TInputControl control is the base class for input controls, and contains all of the core input functionality in the form of public methods and protected properties/events that descendant classes can use to create customized input controls.

Properties	Methods	Events
Error		
InputID		

TInputControl.Error Property

```
property Error: Boolean
```

Specifies whether the input control contains an invalid input value. Setting this property to True will cause the interface state of the control to change to an "Error" state to alert the user that the input value is invalid.

TInputControl.InputID Property

```
property InputID: String
```

Specifies the unique DOM ID to assign to the control's input element. This is useful for situations where you need to identify the element from external Javascript code. By default, Elevate Web Builder never assigns DOM IDs to elements because it doesn't need or use them to identify UI elements.

> **Note**
> This is different from the TControl ClientID property, which is used for accessing the outer client DOM element for a control. An input control typically has both a client element and an input element.

10.123 TInputElement Component

Unit: WebUI

Inherits From TElement

The TInputElement class is the base element class for input elements, and contains all of the input functionality in the form of public methods and properties/events that control classes can use to create input controls.

> **Note**
> This element does not provide support for input elements at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events
AutoComplete	SelectAll	
FieldName	SelectNone	
InputValue		
MaxLength		
Placeholder		
ReadOnly		
SelectionEnd		
SelectionStart		
SpellCheck		

TInputElement.AutoComplete Property

```
property AutoComplete: TAutoCompleteType
```

Specifies how to handle auto-completion for the input element. Auto-completion allows the browser to display a list of suggestions for input values, based upon earlier input values entered by the user.

TInputElement.FieldName Property

```
property FieldName: String
```

Specifies the HTML form value name for the input element. This name is used in conjunction with the InputValue property to generate the HTML form values data used when submitting HTML forms to a web server.

TInputElement.InputValue Property

```
property InputValue: String
```

Specifies the current value of the input element as a string.

TInputElement.MaxLength Property

```
property MaxLength: Integer
```

Specifies the maximum allowable length, in characters, of the InputValue property for the element. A value of 0 specifies an unlimited allowable length.

TInputElement.Placeholder Property

```
property Placeholder: String
```

Specifies a hint to display in the input element before the InputValue property has been assigned a non-blank value.

TInputElement.ReadOnly Property

```
property ReadOnly: Boolean
```

Specifies whether the input element can be modified by the user. Input elements can always be programmatically modified.

TInputElement.SelectionEnd Property

```
property SelectionEnd: Integer
```

Specifies the ending position (1-based) of the selected characters in the input value. For example, if the element contains the input value "Hello World", setting the SelectionStart property to 7 and the SelectionEnd property to 11 will result in the text "World" being selected.

TInputElement.SelectionStart Property

```
property SelectionStart: Integer
```

Specifies the starting position (1-based) of the selected characters in the input value. For example, if the element contains the input value "Hello World", setting the SelectionStart property to 7 and the SelectionEnd property to 11 will result in the text "World" being selected.

TInputElement.SpellCheck Property

```
property SpellCheck: Boolean
```

Specifies whether spell-checking will be enabled for the input element.

TInputElement.SelectAll Method

```
procedure SelectAll
```

Use this method to change the active selection for the input element so that all of the characters in the InputValue property are selected.

TInputElement.SelectNone Method

```
procedure SelectNone
```

Use this method to change the active selection for the input element so that none of the characters in the InputValue property are selected.

10.124 TInsetShadow Component

Unit: WebUI

Inherits From TShadow

The TInsetShadow class represents the inset shadow of a UI element or control. The inset shadow appears within the bounds of the client rectangle for the element.

Properties	Methods	Events

10.125 TIntegerValue Component

Unit: WebCore

Inherits From TDataValue

This class represents the value for an Integer column in a row in a TDataSet component.

Properties	Methods	Events

10.126 TInterface Component

Unit: WebUI

Inherits From TPersistent

The TInterface class represents a control interface that has been loaded by the interface manager at run-time.

Properties	Methods	Events
ControlClassName	Create	
States		

TInterface.ControlClassName Property

```
property ControlClassName: String
```

Specifies the class name of the control that will use the control interface.

TInterface.States Property

```
property States: TInterfaceStates
```

Specifies the various states the make up the control interface.

TInterface.Create Method

```
constructor Create(const AControlClassName: String='')
```

Use this method to create a new instance of the TInterface class. The AControlClassName parameter indicates the class name of the control that will use the control interface.

10.127 TInterfaceController Component

Unit: WebUI

Inherits From TComponent

The TInterfaceController class represents an element's controller. A controller instance is assigned to any element that serves as the base element for a TInterfaceController class descendant.

Note
With the Elevate Web Builder component library, the TInterfaceController class is never instantiated directly. It is used only as a bridge component between the TElement class and the TControl class.

Properties	Methods	Events
	RefreshInterface	
	ResetInterface	

TInterfaceController.RefreshInterface Method

```
procedure RefreshInterface
```

Use this method to refresh the control interface for the controller. Refreshing a control interface causes the current interface state to be re-applied to the controller from a control interface.

TInterfaceController.ResetInterface Method

```
procedure ResetInterface
```

Use this method to reset the control interface for the controller. Resetting a control interface causes the controller to be initialized using the current interface state from a control interface.

10.128 TInterfaceManager Component

Unit: WebUI

Inherits From TObject

The TInterfaceManager class represents the interface manager, which is responsible for managing the user interface of an application at both design-time and run-time. A global instance of the TInterfaceManager class is created automatically at application startup, and is called InterfaceManager.

> **Note**
> Do not create any instances of this class manually, and always use the global instance in order to avoid any conflicts or issues.

Properties	Methods	Events
ActiveElement	ApplyInterface	OnError
ApplicationTitle	BeginAnimation	OnIdle
IdleTimeout	CancelAnimation	OnViewportResize
Interfaces	ContentHeight	OnViewportScroll
IsAndroid	ContentWidth	
IsIOS	ContinueAnimation	
IsWindowsPhone	Create	
RootElement	CreateElement	
ViewportHeight	CreateTimeout	
ViewportResizeDelay	CreateTimer	
ViewportScrollLeft	FreeTimeout	
ViewportScrollTop	FreeTimer	
ViewportWidth	GetInterfaceStateNames	
	GetResource	
	GetResourceCompressed	
	ViewportScroll	

TInterfaceManager.ActiveElement Property

```
property ActiveElement: TElement
```

Indicates the active element in the user interface. The active UI element is the element that currently has focus.

TInterfaceManager.ApplicationTitle Property

```
property ApplicationTitle: String
```

Indicates the title of the application, which is the descriptive name that appears in the web browser for the application's tab or page.

TInterfaceManager.IdleTimeout Property

```
property IdleTimeout: Integer
```

Specifies the time, in seconds, that the application should wait on user input (keypresses, mouse clicks, or touches) before triggering the OnIdle event. This is useful for functionality such as making sure that any authentication information cached for the current user is discarded after a certain period of inactivity, thus forcing the user to login again when interaction with the application is resumed.

Note
Mouse movement alone is not enough to reset the idle timeout. The user must specifically press a key or mouse button, or touch the surface of the screen.

TInterfaceManager.Interfaces Property

```
property Interfaces: TInterfaces
```

Indicates the control interfaces that are loaded by the interface manager at runtime. These control interfaces can be modified by the application at runtime in order to affect the visual appearance of controls.

TInterfaceManager.IsAndroid Property

```
property IsAndroid: Boolean
```

Indicates whether the platform running the application is the Android platform.

TInterfaceManager.IsIOS Property

```
property IsIOS: Boolean
```

Indicates whether the platform running the application is the IOS platform.

TInterfaceManager.IsWindowsPhone Property

```
property IsWindowsPhone: Boolean
```

Indicates whether the platform running the application is the Windows Phone platform.

TInterfaceManager.RootElement Property

```
property RootElement: TElement
```

Indicates the root element of the user interface. At design-time, the root element may be one of two things, depending upon whether the form designer or control interface editor is active:

- If the form designer is active, then the root element will be the base element associated with the active TFormControl instance.
- If the control interface editor is active, then the root element will be the base element for the currently-selected control interface state.

At run-time, the root element is always the base element for the global Application Surface instance.

TInterfaceManager.ViewportHeight Property

```
property ViewportHeight: Integer
```

Indicates the height of the browser viewport.

TInterfaceManager.ViewportResizeDelay Property

```
property ViewportResizeDelay: Integer
```

Specifies how long, in milliseconds, the interface manager will wait after a browser viewport resize before updating the user interface.

TInterfaceManager.ViewportScrollLeft Property

```
property ViewportScrollLeft: Integer
```

Indicates the amount, in pixels, that the browser viewport has been scrolled to the right.

Specify a new value to manually scroll the browser viewport to the left or right. A value of 0 means that the viewport is scrolled all the way to its left-most position.

TInterfaceManager.ViewportScrollTop Property

```
property ViewportScrollTop: Integer
```

Indicates the amount, in pixels, that the browser viewport has been scrolled towards the bottom.

Specify a new value to manually scroll the browser viewport towards the top or bottom. A value of 0 means that the viewport is scrolled all the way to its top-most position.

TInterfaceManager.ViewportWidth Property

```
property ViewportWidth: Integer
```

Indicates the width of the browser viewport.

TInterfaceManager.ApplyInterface Method

```
function ApplyInterface(AElement: TElement; const AClassName:
      String; const AState: String; Initializing: Boolean=False):
      Boolean
```

Use this method to apply a control interface to a base UI element. The AElement parameter specifies the base UI element instance, the AClassName parameter specifies the name of the control interface's class name, the AState parameter specifies the control interface state to apply, and the optional Initializing parameter specifies whether the control interface state should be applied (Initializing=False) or whether the base element should be initialized using the control interface state.

TInterfaceManager.BeginAnimation Method

```
function BeginAnimation(AHandler: TInterfaceAnimationEvent):
      Integer
```

Use this method to add an animation and register an animation frame event handler to be called based upon the monitor refresh rate, if supported, or at 30 times per second if the browser doesn't support the first method.

Animations require that each animation frame event handler execution be scheduled using the BeginAnimation or ContinueAnimation methods, with the BeginAnimation call required at the start of the animation, and the ContinueAnimation call required for all subsequent animation frames until the CancelAnimation method is called.

The return value is an integer that represents the animation, and can be used in further animation operations using the ContinueAnimation and CancelAnimation methods.

Note
This method is not available at design-time. Any calls to it will cause a design-time compilation error when compiling the component library.

TInterfaceManager.CancelAnimation Method

```
procedure CancelAnimation(AID: Integer)
```

Use this method to stop an animation that was started using the BeginAnimation method, or continued using the ContinueAnimation method.

> **Note**
> This method is not available at design-time. Any calls to it will cause a design-time compilation error when compiling the component library.

TInterfaceManager.ContentHeight Method

```
function ContentHeight(AFont: TFont; const AContent: String;
      AHTMLContent: Boolean=False; AAlignment:
      TContentAlignment=caLeft; ADirection:
      TContentDirection=cdLeftToRight; AWrap: Boolean=False;
      AWrapWidth: Integer=0): Integer
```

Use this method to measure the height of text content using the specified font and formatting parameters.

TInterfaceManager.ContentWidth Method

```
function ContentWidth(AFont: TFont; const AContent: String;
      AHTMLContent: Boolean=False; AAlignment:
      TContentAlignment=caLeft; ADirection:
      TContentDirection=cdLeftToRight; AWrap: Boolean=False;
      AWrapWidth: Integer=0): Integer
```

Use this method to measure the width of text content using the specified font and formatting parameters.

TInterfaceManager.ContinueAnimation Method

```
function ContinueAnimation(AID: Integer; AHandler:
      TInterfaceAnimationEvent): Integer
```

Use this method to re-schedule an animation that was started using the BeginAnimation method or already continued using this method.

Animations require that each animation frame event handler execution be scheduled using the BeginAnimation or ContinueAnimation methods, with the BeginAnimation call required at the start of the animation, and the ContinueAnimation call required for all subsequent animation frames until the CancelAnimation method is called.

Warning

This method should be called from within the animation event handler specified by the last BeginAnimation or ContinueAnimation call.

Note

This method is not available at design-time. Any calls to it will cause a design-time compilation error when compiling the component library.

TInterfaceManager.Create Method

```
constructor Create
```

Use this method to create a new instance of the TInterfaceManager class.

TInterfaceManager.CreateElement Method

```
function CreateElement(const AName: String; AParent:
      TElement=nil; const AClassName: String=ELEMENT_CLASS_DIV;
      AContainer: Boolean=False; AEvents: Boolean=False; ADynamic:
      Boolean=False): TElement
```

Use this method to create a new UI element.

The AName parameter specifies the name of the element.

The optional AParent parameter specifies the parent element, if any.

The optional AClassName parameter specifies the element class name to use when creating the element.

The optional AContainer parameter specifes whether or not the element is a container. A container element is one that is capable of being a container for other elements at design-time.

The optional AEvents parameter specifies whether the element wants design-time mouse events in order to allow the developer to interact with the element at design-time.

The optional ADynamic parameter specifies whether the element is being instantiated dynamically at design-time via interaction by the developer.

TInterfaceManager.CreateTimeout Method

```
function CreateTimeout(AHandler: TInterfaceTimeoutEvent;
      AInterval: Integer): Integer
```

Use this method to create a timeout event. The AHandler parameter specifies the event handler to call when the AInterval milliseconds parameter has elapsed.

TInterfaceManager.CreateTimer Method

```
function CreateTimer(AHandler: TInterfaceTimerEvent; AInterval:
      Integer): Integer
```

Use this method to create a timer event. The AHandler parameter specifies the event handler to call every time the AInterval milliseconds parameter has elapsed.

TInterfaceManager.FreeTimeout Method

```
procedure FreeTimeout(AID: Integer)
```

Use this method to free a timeout event.

TInterfaceManager.FreeTimer Method

```
procedure FreeTimer(AID: Integer)
```

Use this method to free a timer event.

TInterfaceManager.GetInterfaceStateNames Method

```
function GetInterfaceStateNames(const AClassName: String): array
      of String
```

Use this method to retrieve a list of defined state names as an array of strings. The AClassName is the control interface class name for which you want the list of state names.

Note
This method is used by the icon library in Elevate Web Builder to retrieve the list of icons defined in the icon library.

TInterfaceManager.GetResource Method

```
function GetResource(const ResourceType: String; const
      ResourceName: String): String
```

Use this method to retrieve an embedded resource for an application at run-time.

> **Note**
> This method is not available at design-time. Any calls to it will cause a design-time compilation error when compiling the component library.

TInterfaceManager.GetResourceCompressed Method

```
function GetResourceCompressed(const ResourceType: String; const
      ResourceName: String): Boolean
```

Use this method to retrieve a compressed and embedded resource for an application at run-time.

Note
This method is not available at design-time. Any calls to it will cause a design-time compilation error when compiling the component library.

TInterfaceManager.ViewportScroll Method

```
procedure ViewportScroll(X,Y: Integer)
```

Use this method to manually scroll the browser viewport at run-time.

Note
This method is not available at design-time. Any calls to it will cause a design-time compilation error when compiling the component library.

TInterfaceManager.OnError Event

```
property OnError: TInterfaceErrorEvent
```

This event is triggered when any unhandled exception occurs in the application at runtime in the browser.

Note
Do not set an event handler for this event. The global TApplication instance for visual applications assigns an event handler for this event.

TInterfaceManager.OnIdle Event

```
property OnIdle: TInterfaceIdleEvent
```

This event is triggered when the application's IdleTimeout property has been exceeded.

Note
Do not set an event handler for this event. The global TApplication instance for visual applications assigns an event handler for this event.

TInterfaceManager.OnViewportResize Event

```
property OnViewportResize: TInterfaceViewportResizeEvent
```

This event is triggered whenever the browser viewport's width and/or height are changed.

TInterfaceManager.OnViewportScroll Event

```
property OnViewportScroll: TInterfaceViewportScrollEvent
```

This event is triggered whenever the browser viewport is scrolled horizontally or vertically.

10.129 TInterfaces Component

Unit: WebUI

Inherits From TObject

The TInterfaces class represents the control interfaces that have been loaded by the interface manager at run-time.

Note
You can modify the control interfaces after they have been loaded at runtime in order to customize how various controls look.

Properties	Methods	Events
Count	Create	
Interfaces	FindInterfaceByClassName	
	Load	
	RemoveAll	

TInterfaces.Count Property

```
property Count: Integer
```

Indicates the total number of defined control interfaces.

TInterfaces.Interfaces Property

```
property Interfaces[Index: Integer]: TInterface
```

Accesses a control interface by its index position in the defined control interfaces.

TInterfaces.Create Method

```
constructor Create
```

Use this method to create a new instance of the TInterfaces class.

TInterfaces.FindInterfaceByClassName Method

```
function FindInterfaceByClassName(const AClassName: String):
      TInterface
```

Use this method to find a control interface by its control class name. If the control interface does not exist, then nil will be returned.

TInterfaces.Load Method

```
procedure Load
```

Use this method to load all control interfaces bundled with the visual application from the HTML loader file for the application. This method is automatically called during application startup and should not be called manually.

TInterfaces.RemoveAll Method

```
procedure RemoveAll
```

Use this method to remove all loaded control interfaces. It is not advised to use this method because doing so can cause a visual application to lose all interface attributes.

10.130 TInterfaceState Component

Unit: WebUI

Inherits From TPersistent

The TInterfaceState class represents a state defined for a control interface that has been loaded by the interface manager at run-time.

Properties	Methods	Events
Name		
RootElement		

TInterfaceState.Name Property

```
property Name: String
```

Specifies the name of the control interface state.

TInterfaceState.RootElement Property

```
property RootElement: TElement
```

Indicates the root element for the control interface state.

10.131 TInterfaceStates Component

Unit: WebUI

Inherits From TPersistent

The TInterfaceStates class represents the states defined for a control interface that has been loaded by the interface manager at run-time.

Properties	Methods	Events
Count	Create	
DefaultState	FindState	
State	NewState	
	RemoveState	
	RenameState	

TInterfaceStates.Count Property

```
property Count: Integer
```

Indicates the total number of defined control interface states.

TInterfaceStates.DefaultState Property

```
property DefaultState: TInterfaceState
```

Indicates the default control interface state, which corresponds to the first defined control interface state.

TInterfaceStates.State Property

```
property State[Index: Integer]: TInterfaceState
```

Accesses a control interface state by its index position in the defined control interface states.

TInterfaceStates.Create Method

```
constructor Create(AInterface: TInterface)
```

Use this method to create a new instance of the TInterfaceStates class. The AInterface parameter indicates the parent interface instance that will manage the states.

TInterfaceStates.FindState Method

```
function FindState(const AName: String): TInterfaceState
```

Use this method to find a control interface state by its name. If the state does not exist, then nil will be returned.

TInterfaceStates.NewState Method

```
function NewState(const AName: String; const FromName:
      String=''): TInterfaceState
```

Use this method to create a new control interface state. The AName parameter indicates the name of the new state, and the optional FromName parameter indicates the name of an existing state that will be used to initialize the new state. If the state indicated by the AName parameter already exists, then an exception will be raised.

TInterfaceStates.RemoveState Method

```
function RemoveState(const AName: String): Boolean
```

Use this method to remove a control interface state. If the state does not exist, then False will be returned.

TInterfaceStates.RenameState Method

```
function RenameState(const AName: String; const NewName:
      String): Boolean
```

Use this method to rename a control interface state. If the state does not exist, then False will be returned.

10.132 TLabel Component

Unit: WebLabels

Inherits From TLabelControl

The TLabel component represents a label control. A label control displays text content using a specific font and formatting, including alignment and wrapping.

Properties	Methods	Events
AllowCopy		OnAnimationComplete
AutoSize		OnAnimationsComplete
Background		OnClick
Border		OnDblClick
Caption		OnHide
Corners		OnMouseDown
Cursor		OnMouseEnter
DataColumn		OnMouseLeave
DataSet		OnMouseMove
FocusControl		OnMouseUp
Font		OnMove
Format		OnShow
Hint		OnSize
Opacity		OnTouchCancel
OutsetShadow		OnTouchEnd
Padding		OnTouchMove
		OnTouchStart

TLabel.AllowCopy Property

```
property AllowCopy: Boolean
```

Specifies whether the label control will allow its contents to be copied by the user.

Note
You must also set the Cursor property to crText if you want the user to be able to directly copy the contents of the label using the mouse or touch.

TLabel.AutoSize Property

```
property AutoSize: TAutoSize
```

Specifies how (if at all) the control should automatically be sized based upon the Caption and Format properties.

TLabel.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TLabel.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TLabel.Caption Property

```
property Caption: String
```

Specifies the textual caption to display in the control. The default value is ''.

TLabel.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TLabel.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TLabel.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TLabel.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TLabel.FocusControl Property

```
property FocusControl: TInputControl
```

Specifies a control that will receive focus when the label is clicked.

TLabel.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TLabel.Format Property

```
property Format: TFormat
```

Specifies the content formatting to use for the control's Caption.

TLabel.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TLabel.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TLabel.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the control.

TLabel.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

TLabel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TLabel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TLabel.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TLabel.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TLabel.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TLabel.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TLabel.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TLabel.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TLabel.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TLabel.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TLabel.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TLabel.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TLabel.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TLabel.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TLabel.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TLabel.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TLabel.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.133 TLabelControl Component

Unit: WebLabels

Inherits From TBindableColumnControl

The TLabelControl control is the base class for label controls, and contains all of the core label functionality in the form of public methods and protected properties/events that descendant classes can use to create customized label controls.

Properties	Methods	Events

10.134 TLayout Component

Unit: WebUI

Inherits From TElementAttribute

The TLayout class represents the layout properties of a UI element or control. The layout properties include positioning, stretching, and layout space consumption. Please see the Layout Management topic for more information on how the layout properties are used.

Properties	Methods	Events
Consumption	SetToDefault	
Overflow		
Position		
Reset		
Stretch		

TLayout.Consumption Property

```
property Consumption: TLayoutConsumption
```

Specifies how the UI element or control consumes space in the current layout rectangle, if at all.

TLayout.Overflow Property

```
property Overflow: TLayoutOverflow
```

Specifies the direction in which the current UI element or control should reset the consumption of the prior UI element or control when the current UI element or control will not fit within the bounds of the current layout rectangle.

TLayout.Position Property

```
property Position: TLayoutPosition
```

Specifies how the UI element or control is positioned in the current layout rectangle, if at all.

TLayout.Reset Property

```
property Reset: Boolean
```

Specifies whether the UI element or control is resetting the space consumption direction in the current layout rectangle.

TLayout.Stretch Property

```
property Stretch: TLayoutStretch
```

Specifies how the UI element or control is stretched in the current layout rectangle, if at all.

TLayout.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the layout's properties to their default values.

10.135 TLink Component

Unit: WebBrwsr

Inherits From TLinkControl

The TLink component represents a link control. A link control displays link text using the Caption property that results in navigation to the specified URL property when the control is clicked at run-time.

> **Note**
> This control does not provide support for links at design-time.

Properties	Methods	Events
AutoSize		OnAnimationComplete
Background		OnAnimationsComplete
Border		OnClick
Caption		OnHide
Corners		OnMouseDown
Cursor		OnMouseEnter
DataColumn		OnMouseLeave
DataSet		OnMouseMove
Font		OnMouseUp
Format		OnMove
Hint		OnShow
NewWindow		OnSize
Opacity		OnTouchCancel
OutsetShadow		OnTouchEnd
Padding		OnTouchMove
TabOrder		OnTouchStart
TabStop		
URL		

TLink.AutoSize Property

```
property AutoSize: TAutoSize
```

Specifies how (if at all) the control should automatically be sized based upon the Caption and Format properties.

TLink.Background Property

```
property Background: TBackground
```

Specifies the background for the control.

TLink.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TLink.Caption Property

```
property Caption: TCaption
```

Specifies the textual caption to display in the control. The default value is ''.

TLink.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TLink.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TLink.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TLink.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TLink.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TLink.Format Property

```
property Format: TFormat
```

Specifies the content formatting to use for the control's Caption.

TLink.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TLink.NewWindow Property

```
property NewWindow: Boolean
```

Specifies that the resource represented by the URL property should be opened in a new browser window or page when the link control is clicked.

TLink.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TLink.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the control.

TLink.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

TLink.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TLink.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TLink.URL Property

```
property URL: String
```

Specifies the URL for the resource that is the target of the link.

TLink.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TLink.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TLink.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TLink.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TLink.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TLink.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TLink.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TLink.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TLink.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TLink.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TLink.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TLink.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TLink.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TLink.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TLink.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TLink.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.136 TLinkControl Component

Unit: WebBrwsr

Inherits From TBindableColumnControl

The TLinkControl control is the base class for link controls, and contains all of the core link functionality in the form of public methods and protected properties/events that descendant classes can use to create customized link controls.

Properties	Methods	Events

10.137 TLinkElement Component

Unit: WebUI

Inherits From TElement

The TLinkElement class is the element class for URL links, and contains all of the URL link functionality in the form of public methods and properties/events that control classes can use to create link controls.

Note
This element does not provide support for links at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events
NewWindow	OpenURL	
URL		

TLinkElement.NewWindow Property

```
property NewWindow: Boolean
```

Specifies that the navigation that occurs in response to a click or selection should cause a new browser window to open.

TLinkElement.URL Property

```
property URL: String
```

Specifies the URL of the link.

TLinkElement.OpenURL Method

```
procedure OpenURL
```

Use this method to programmatically open the link specified in the URL property. If the NewWindow property is True, then the URL is opened in a new browser window or tab.

10.138 TListBox Component

Unit: WebLists

Inherits From TListControl

The TListBox component represents a listbox control for displaying a list of selectable items specified by the Items property. Multiple items can be selected if the MultiSelect property is set to True.

Note
This control is a virtual control, meaning that it can store and display very large numbers of items efficiently by only using UI elements for the visible items in the control.

Properties	Methods	Events
AutoItemHeight		OnAnimationComplete
Corners		OnAnimationsComplete
Cursor		OnChange
DataColumn		OnClick
DataSet		OnDblClick
Enabled		OnEnter
Font		OnExit
Hint		OnHide
ItemHeight		OnKeyDown
ItemIndex		OnKeyPress
Items		OnKeyUp
KeyPressInterval		OnMouseDown
MultiSelect		OnMouseEnter
ReadOnly		OnMouseLeave
ScrollBar		OnMouseMove
ScrollSupport		OnMouseUp
Selected		OnMouseWheel
SelectedCount		OnMove
Sorted		OnScroll
TabOrder		OnShow
TabStop		OnSize
Text		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchScroll
		OnTouchStart

TListBox.AutoItemHeight Property

```
property AutoItemHeight: Boolean
```

Specifies that the displayed height of the items will automatically be set based upon the Font property settings. The default value is True.

TListBox.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TListBox.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TListBox.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TListBox.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TListBox.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TListBox.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TListBox.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TListBox.ItemHeight Property

```
property ItemHeight: Integer
```

Specifies the height, in pixels, of the items displayed in the list control.

TListBox.ItemIndex Property

```
property ItemIndex: Integer
```

Indicates the index of the currently-focused item in the list control, or -1 if no item is currently focused.

TListBox.Items Property

```
property Items: TStrings
```

Specifies the items to display in the list control.

TListBox.KeyPressInterval Property

```
property KeyPressInterval: Integer
```

Specifies the interval, in milliseconds, that is used by the control to combine user keystrokes into a search value that is then used for performing a near search on the Items property. Effectively, this means that the user has KeyPressInterval milliseconds in which to hit a key in order for the keystroke to be included as part of a near search. The default value is 300 milliseconds.

For example, if the user hits the 'S', 'M', and 'I' keys within the KeyPressInterval property value, but hits the 'T' key outside of the KeyPressInterval property, then the control will perform a near search using the value 'SMI', followed by a near search using the value 'T'.

TListBox.MultiSelect Property

```
property MultiSelect: Boolean
```

Specifies whether multiple items may be selected. If this property is True, then the user can select multiple items in the list control using an individual selection operation (Ctrl-Click), or using a range selection operation (Shift-Click). In addition, the developer can directly modify the Selected property to select or deselect any item(s) in the list control.

The default value is False.

TListBox.ReadOnly Property

```
property ReadOnly: Boolean
```

Specifies whether the listbox's selected item(s) can be modified by the user. The default value is False.

Note
The listbox's selected item(s) can always be programmatically modified.

TListBox.ScrollBar Property

```
property ScrollBar: Boolean
```

Specifies whether the vertical scrollbar should be shown for the listbox control.

Note
Even if this property is set to True, a vertical scrollbar will only be shown if the vertical size of the contents of the control exceed the client rectangle for the control.

TListBox.ScrollSupport Property

```
property ScrollSupport: Boolean
```

Specifies whether to allow vertical scrolling in the listbox control.

Note
This property only applies to scrolling via touch and mouse wheel movements, and does not apply to programmatic scrolling or scrolling via the vertical scroll bar itself.

TListBox.Selected Property

```
property Selected[AIndex: Integer]: Boolean
```

Specifies which items in the list control are currently selected. If the MultiSelect property is True, then this property can be used to test which items are selected by their index, or to select one or more items. If the MultiSelect property is False, then only one item at a time will be indicated as being selected in this property, and you will not be able to modify the selection status of items.

TListBox.SelectedCount Property

```
property SelectedCount: Integer
```

Specifies the number of selected items in the list control. If the MultiSelect property is True, then this property can be used to find out how many items are selected. If the MultiSelect property is False, then this property will always be equal to 1 or 0 (if the list is empty).

TListBox.Sorted Property

```
property Sorted: Boolean
```

Specifies whether the list items will automatically be sorted. The default value is False.

TListBox.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TListBox.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TListBox.Text Property

```
property Text: String
```

Specifies the list control's selected items(s) as a string.

TListBox.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TListBox.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TListBox.OnChange Event

```
property OnChange: TNotifyEvent
```

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TListBox.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TListBox.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TListBox.OnEnter Event

```
property OnEnter: TNotifyEvent
```

This event is triggered when the control obtains input focus.

TListBox.OnExit Event

```
property OnExit: TNotifyEvent
```

This event is triggered when the control loses input focus.

TListBox.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TListBox.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when the control has input focus and the user presses a key or key combination.

TListBox.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TListBox.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when the control has input focus and the user releases a key or key combination.

TListBox.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TListBox.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TListBox.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TListBox.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TListBox.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TListBox.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

This event is triggered whenever the mouse wheel is rotated forward or backward.

TListBox.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TListBox.OnScroll Event

```
property OnScroll: TNotifyEvent
```

This event is triggered whenever a scrollable control is scrolled horizontally or vertically.

TListBox.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TListBox.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TListBox.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TListBox.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TListBox.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TListBox.OnTouchScroll Event

```
property OnTouchScroll: TTouchScrollEvent
```

This event is triggered whenever a touch moves in any direction over a touch-scroll-enabled control.

TListBox.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.139 TListControl Component

Unit: WebLists

Inherits From TInputControl

The TListControl control is the base class for list controls, and contains all of the core list functionality in the form of public methods and protected properties/events that descendant classes can use to create customized list controls.

Properties	Methods	Events
	SelectAll	
	SelectRange	
	ToggleSelected	

TListControl.SelectAll Method

```
procedure SelectAll
```

Use this method to select all items in the list control.

TListControl.SelectRange Method

```
procedure SelectRange(AFromIndex, AToIndex: Integer; AClear:
      Boolean=False)
```

Use this method to select a range of items in the list control. The AClear parameter determines whether the existing set of selected items should be cleared before the new range of items is selected.

TListControl.ToggleSelected Method

```
procedure ToggleSelected(AIndex: Integer)
```

Use this method to toggle the selection state of the item at the specified index.

10.140 TLocation Component

Unit: WebComps

Inherits From TObject

The TLocation class represents current location information for the global LocationServices instance of the TLocationServices class and contains properties for all available location information.

Properties	Methods	Events
Accuracy		
Altitude		
AltitudeAccuracy		
AltitudeAvailable		
Heading		
HeadingAvailable		
Latitude		
Longitude		
Speed		
SpeedAvailable		
Timestamp		

TLocation.Accuracy Property

```
property Accuracy: Double
```

Specifies the accuracy, in meters, of the Latitude and Longitude properties.

TLocation.Altitude Property

```
property Altitude: Double
```

Specifies the altitude, in meters, relative to sea level.

Note
This property is only valid when the AltitudeAvailable property is True.

TLocation.AltitudeAccuracy Property

```
property AltitudeAccuracy: Double
```

Specifies the accuracy, in meters, of the Altitude property.

Note
This property is only valid when the AltitudeAvailable property is True.

TLocation.AltitudeAvailable Property

```
property AltitudeAvailable: Boolean
```

Specifies whether altitude information was available as part of the returned location information.

Note
The Altitude and AltitudeAccuracy properties are only valid when the AltitudeAvailable property is True.

TLocation.Heading Property

```
property Heading: Double
```

Specifies the heading, in degrees, relative to true north. East is 90 degrees, south is 180 degrees, and west is 270 degrees. If the Speed property is 0, then this property will be 0.

Note
This property is only valid when the HeadingAvailable property is True.

TLocation.HeadingAvailable Property

```
property HeadingAvailable: Boolean
```

Specifies whether heading information was available as part of the returned location information.

Note
The Heading property is only valid when the HeadingAvailable property is True.

TLocation.Latitude Property

```
property Latitude: Double
```

Specifies the latitude, in meters.

TLocation.Longitude Property

```
property Longitude: Double
```

Specifies the longitude, in meters.

TLocation.Speed Property

```
property Speed: Double
```

Specifies the velocity, in meters per second.

Note
This property is only valid when the SpeedAvailable property is True.

TLocation.SpeedAvailable Property

```
property SpeedAvailable: Boolean
```

Specifies whether speed information was available as part of the returned location information.

Note
The Speed property is only valid when the SpeedAvailable property is True.

TLocation.Timestamp Property

```
property Timestamp: DateTime
```

Specifies the timestamp of when the location information was obtained.

10.141 TLocationServices Component

Unit: WebComps

Inherits From TObject

The TLocationServices object encapsulates the HTML5 geolocation functionality, which allows the application to determine the physical location of the machine or device running the host web browser (with the user's permission). In addition to the latitude and longitude of the machine or device, additional information such as the altitude, heading, and speed may be available, depending upon the type of device being used to host the web browser.

Note
The component library includes one global instance of this class called LocationServices in the WebComps unit that should be used instead of creating a new instance of the class.

Warning
The HTML5 geolocation functionality is only available in secure contexts (https) in many modern browsers, do please keep this in mind when deciding whether to use such functionality in your application.

Properties	Methods	Events
Error	Create	OnLocationError
HighAccuracy	StartTrackingLocation	OnLocationUpdate
Location	StopTrackingLocation	
MaxAge	UpdateLocation	
Timeout		

TLocationServices.Error Property

```
property Error: TLocationError
```

Specifies the error condition when the current location cannot be obtained.

Note
This property is only valid after the OnLocationError or OnLocationUpdate event handlers have been executed.

TLocationServices.HighAccuracy Property

```
property HighAccuracy: Boolean
```

Specifies that the next attempt to obtain the current location information via the UpdateLocation or StartTrackingLocation methods should request a high level of accuracy from the machine or device hosting the web browser. If the machine or device is able to do so, it will use the most accurate method at its disposal to provide the location information.

TLocationServices.Location Property

```
property Location: TLocation
```

Note
This property is only valid after the OnLocationError or OnLocationUpdate event handlers have been executed.

TLocationServices.MaxAge Property

```
property MaxAge: Integer
```

Specifies that the next attempt to obtain the current location information via the UpdateLocation or StartTrackingLocation methods should request that any returned location information not have been cached for longer than the specified number of milliseconds. If this property is set to 0 (the default value), any returned location information will not be cached information. If this property is set to -1, any returned location information will only be cached information, no matter how old the information is.

TLocationServices.Timeout Property

```
property Timeout: Integer
```

Specifies that the next attempt to obtain the current location information via the UpdateLocation or StartTrackingLocation methods should request that any returned location information must be returned within a certain number of milliseconds. If this property is set to 0 (the default value), then the location information must be returned immediately or the OnLocationError event handler will be executed. If this property is set to -1, then the OnLocationError event handler will never be executed due to a timeout, and the OnLocationUpdate event handler will not be executed until the location information is available, no matter how long it takes.

TLocationServices.Create Method

```
constructor Create
```

Use this method to create a new instance of the TLocationServices class.

TLocationServices.StartTrackingLocation Method

```
procedure StartTrackingLocation
```

Use this method to start tracking the location information for the machine or device using the HighAccuracy, MaxAge, and Timeout properties to control how the location information is obtained.

After the tracking is started, the OnLocationUpdate event handler will be executed any time the location information changes and the Location property will contain the location information. If the location information cannot be obtained for any reason, then the OnLocationError event handler will be executed and the Error property will contain the error condition.

TLocationServices.StopTrackingLocation Method

```
procedure StopTrackingLocation
```

Use this method to stop tracking the location information for the machine or device.

TLocationServices.UpdateLocation Method

```
procedure UpdateLocation
```

Use this method to obtain location information for the machine or device using the HighAccuracy, MaxAge, and Timeout properties to control how the location information is obtained.

If the location information is successfully obtained, the OnLocationUpdate event handler will be executed and the Location property will contain the location information. If the location information cannot be obtained for any reason, then the OnLocationError event handler will be executed and the Error property will contain the error condition.

TLocationServices.OnLocationError Event

```
property OnLocationError: TNotifyEvent
```

This event is triggered when an attempt to obtain the location information via the UpdateLocation or StartTrackingLocation methods fails. Once this event is triggered, the Error property will contain the error condition.

TLocationServices.OnLocationUpdate Event

```
property OnLocationUpdate: TNotifyEvent
```

This event is triggered when an attempt to obtain the location information via the UpdateLocation or StartTrackingLocation methods is successful. Once this event is triggered, the Location property will contain the location information.

10.142 TMap Component

Unit: WebMaps

Inherits From TMapControl

The TMap component represents a map control that loads and uses the Google Maps API for map display and geocoding.

Properties	Methods	Events
APIKey		OnAnimationComplete
APIKeyRequired		OnAnimationsComplete
Background		OnAPIError
Border		OnAPILoad
Corners		OnHide
Cursor		OnMove
InsetShadow		OnShow
Locations		OnSize
Options		
OutsetShadow		
Padding		

TMap.APIKey Property

```
property APIKey: String
```

Google has changed the requirements for using the Google Maps API:

Google Maps Standard Plan Updates

and the TMap control now includes two new properties to accomodate the new API key requirements: APIKeyRequired and APIKey. If you were using the Google Maps API successfully with your application prior to the new requirements, then your IP address will be grandfathered in and you can set the APIKeyRequired property to False and leave the APIKey property blank.

TMap.APIKeyRequired Property

```
property APIKeyRequired: Boolean
```

Google has changed the requirements for using the Google Maps API:

Google Maps Standard Plan Updates

and the TMap control now includes two new properties to accomodate the new API key requirements: APIKeyRequired and APIKey. If you were using the Google Maps API successfully with your application prior to the new requirements, then your IP address will be grandfathered in and you can set the APIKeyRequired property to False and leave the APIKey property blank.

TMap.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TMap.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TMap.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TMap.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TMap.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Specifies the inset shadow for the control.

TMap.Locations Property

```
property Locations: TMapLocations
```

Provides access to the defined locations for the map.

TMap.Options Property

```
property Options: TMapOptions
```

Specifies the map options.

TMap.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the control.

TMap.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

TMap.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TMap.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TMap.OnAPIError Event

```
property OnAPIError: TNotifyEvent
```

This event is triggered when the Google Maps API cannot be loaded due to an error condition.

TMap.OnAPILoad Event

```
property OnAPILoad: TNotifyEvent
```

This event is triggered when the Google Maps API has been completely loaded and is ready for use.

Note
The Google Maps API is only loaded once and used with every TMap instance, but this event will be triggered for each TMap instance even if the API has already been loaded.

TMap.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TMap.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TMap.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TMap.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

10.143 TMapControl Component

Unit: WebMaps

Inherits From TControl

The TMapControl control is the base class for map controls that dynamically load and use the Google Maps API. It contains all of the map control functionality in the form of public methods and protected properties/events that descendant classes can use to create customized map controls.

Properties	Methods	Events

10.144 TMapLocation Component

Unit: WebMaps

Inherits From TCollectionItem

The TMapLocation class represents a named location in a TMap control and contains functionality for getting and setting an address or latitude/longitude, as well as placing markers on the map or changing the map so that the location is at the center.

Properties	Methods	Events
Address		OnClick
Center		
Icon		
Latitude		
Longitude		
ShowMarker		
Title		

TMapLocation.Address Property

```
property Address: String
```

Specifies an address for the named location. When this property is changed, the Latitude and Longitude properties are set to 0 and the address is geocoded by Google Maps. If the geocoding is successful, then:

- If the Center property is set to True, the location will be centered on the map.
- If the SetMarker property is set to True, the a marker will be placed on the location on the map, and the Title will be displayed as a hint when the mouse hovers over the marker.

TMapLocation.Center Property

```
property Center: Boolean
```

Specifies whether the location should be set as the center of the map. The default value is False.

Note
Only one named location on the map can have its Center property set to True, and the TMap control will enforce this rule by setting the Center property to False for any other defined locations.

TMapLocation.Icon Property

```
property Icon: String
```

Specifies the URL for an icon to use in place of the standard location marker.

TMapLocation.Latitude Property

```
property Latitude: Double
```

Specifies the latitude for the named location. When this property is changed, the Address property is set to ''. If both the Latitude and Longitude properties are assigned non-zero values, then the latitude and longitude are assigned to the map, and:

- If the Center property is set to True, the location will be centered on the map.
- If the SetMarker property is set to True, the a marker will be placed on the location on the map, and the Title will be displayed as a hint when the mouse hovers over the marker.

TMapLocation.Longitude Property

```
property Longitude: Double
```

Specifies the longitude for the named location. When this property is changed, the Address property is set to ''. If both the Latitude and Longitude properties are assigned non-zero values, then the latitude and longitude are assigned to the map, and:

- If the Center property is set to True, the location will be centered on the map.
- If the SetMarker property is set to True, the a marker will be placed on the location on the map, and the Title will be displayed as a hint when the mouse hovers over the marker.

TMapLocation.ShowMarker Property

```
property ShowMarker: Boolean
```

Specifies whether a marker should be set for the named location on the map. The default value is False.

TMapLocation.Title Property

```
property Title: String
```

Specifies the title for the named location to show when the mouse hovers over the marker. This property is only valid when the SetMarker property is set to True.

TMapLocation.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

10.145 TMapLocations Component

Unit: WebMaps

Inherits From TCollection

The TMapLocations class represents the list of named locations in a TMap control and contains functionality for managing the locations.

Properties	Methods	Events
Location		

TMapLocations.Location Property

```
property Location[Index: Integer]: TMapLocation

property Location[const Name: String]: TMapLocation
```

Accesses all locations defined for the map by index or by name.

10.146 TMapOption Component

Unit: WebMaps

Inherits From TPersistent

The TMapOption class is the base class for any complex TMap control mapping options.

Properties	Methods	Events
	Create	

TMapOption.Create Method

```
constructor Create(AMap: TMapControl; AParent: TMapOption)
```

Use this method to create a new instance of the TMapOption class. The AMap parameter indicates the map control instance that will manage the option, and the AParent parameter indicates the parent option, if any, that the option is contained within. The parent option is used to aggregate change management at the outermost option so as to avoid excessively triggering change notifications in the map control.

10.147 TMapOptions Component

Unit: WebMaps

Inherits From TMapOption

The TMapOptions class represents a list of map options for a TMap control. These map options correspond to the map options available for maps in the Google Maps API.

Properties	Methods	Events
DisableDblClickZoom		
DisableDefaultUI		
Draggable		
Heading		
KeyboardShortCuts		
MapType		
MapTypeControl		
MapTypeControlOptions		
MaxZoom		
MinZoom		
OverviewMapControl		
OverviewMapControlOptions		
PanControl		
PanControlOptions		
RotateControl		
RotateControlOptions		
ScaleControl		
ScrollWheel		
StreetViewControl		
StreetViewControlOptions		
Tilt		
Zoom		
ZoomControl		
ZoomControlOptions		

TMapOptions.DisableDblClickZoom Property

```
property DisableDblClickZoom: Boolean
```

Enables/disables zoom and center on double click. Enabled by default.

TMapOptions.DisableDefaultUI Property

```
property DisableDefaultUI: Boolean
```

Enables/disables all default UI. May be overridden individually.

TMapOptions.Draggable Property

```
property Draggable: Boolean
```

If False, prevents the map from being dragged. Dragging is enabled by default.

TMapOptions.Heading Property

```
property Heading: Double
```

The heading for aerial imagery in degrees measured clockwise from cardinal direction North. Headings are snapped to the nearest available angle for which imagery is available.

TMapOptions.KeyboardShortCuts Property

```
property KeyboardShortCuts: Boolean
```

If false, prevents the map from being controlled by the keyboard. Keyboard shortcuts are enabled by default.

TMapOptions.MapType Property

```
property MapType: TMapType
```

Specifies the initial map type. The default value is mtRoadmap.

TMapOptions.MapTypeControl Property

```
property MapTypeControl: Boolean
```

The initial enabled/disabled state of the map type control.

TMapOptions.MapTypeControlOptions Property

```
property MapTypeControlOptions: TMapTypeControlOptions
```

The initial display options for the map type control.

TMapOptions.MaxZoom Property

```
property MaxZoom: Integer
```

The maximum zoom level which will be displayed on the map.

TMapOptions.MinZoom Property

```
property MinZoom: Integer
```

The minimum zoom level which will be displayed on the map.

TMapOptions.OverviewMapControl Property

```
property OverviewMapControl: Boolean
```

The enabled/disabled state of the overview map control.

TMapOptions.OverviewMapControlOptions Property

```
property OverviewMapControlOptions: TOverviewMapControlOptions
```

The display options for the overview map control.

TMapOptions.PanControl Property

```
property PanControl: Boolean
```

The enabled/disabled state of the pan control.

TMapOptions.PanControlOptions Property

```
property PanControlOptions: TPanControlOptions
```

The display options for the pan control.

TMapOptions.RotateControl Property

```
property RotateControl: Boolean
```

The enabled/disabled state of the rotate control.

TMapOptions.RotateControlOptions Property

```
property RotateControlOptions: TRotateControlOptions
```

The display options for the rotate control.

TMapOptions.ScaleControl Property

```
property ScaleControl: Boolean
```

The initial enabled/disabled state of the scale control.

TMapOptions.ScrollWheel Property

```
property ScrollWheel: Boolean
```

If false, disables mouse scrollwheel zooming on the map. The mouse scrollwheel is enabled by default.

TMapOptions.StreetViewControl Property

```
property StreetViewControl: Boolean
```

The initial enabled/disabled state of the street view pegman control. This control is part of the default UI, and should be set to false when displaying a map type on which the street view road overlay should not appear (e.g. a non-Earth map type).

TMapOptions.StreetViewControlOptions Property

```
property StreetViewControlOptions: TStreetViewControlOptions
```

The initial display options for the street view pegman control.

TMapOptions.Tilt Property

```
property Tilt: TMapTilt
```

Controls the automatic switching behavior for the angle of incidence of the map. The only allowed values are 0 and 45. The value 0 causes the map to always use a 0° overhead view regardless of the zoom level and viewport. The value 45 causes the tilt angle to automatically switch to 45 whenever 45° imagery is available for the current zoom level and viewport, and switch back to 0 whenever 45° imagery is not available (this is the default behavior). 45° imagery is only available for the mtSatellite and mtHybrid map types, within some locations, and at some zoom levels.

TMapOptions.Zoom Property

```
property Zoom: Integer
```

The initial map zoom level.

TMapOptions.ZoomControl Property

```
property ZoomControl: Boolean
```

The enabled/disabled state of the zoom control.

TMapOptions.ZoomControlOptions Property

```
property ZoomControlOptions: TZoomControlOptions
```

The display options for the zoom control.

10.148 TMapTypeControlMapTypes Component

Unit: WebMaps

Inherits From TMapOption

The TMapTypeControlMapTypes class represents the list of map types that should be available for user selection using the map type control in a TMap control. These map types correspond to the map types available for maps in the Google Maps API.

Properties	Methods	Events
Hybrid		
Roadmap		
Satellite		
Terrain		

TMapTypeControlMapTypes.Hybrid Property

```
property Hybrid: Boolean
```

Include hybrid as a selectable map type.

TMapTypeControlMapTypes.Roadmap Property

```
property Roadmap: Boolean
```

Include roadmap as a selectable map type.

TMapTypeControlMapTypes.Satellite Property

```
property Satellite: Boolean
```

Include satellite as a selectable map type.

TMapTypeControlMapTypes.Terrain Property

```
property Terrain: Boolean
```

Include terrain as a selectable map type.

10.149 TMapTypeControlOptions Component

Unit: WebMaps

Inherits From TMapOption

The TMapTypeControlOptions class controls how the map type control is configured in a TMap control. These map type control options correspond to the map type control options available for maps in the Google Maps API.

Properties	Methods	Events
MapTypes		
Position		
Style		

TMapTypeControlOptions.MapTypes Property

```
property MapTypes: TMapTypeControlMapTypes
```

Specifies which map types to show in the map type control.

TMapTypeControlOptions.Position Property

```
property Position: TMapControlPosition
```

Specifies the position of the map type control.

TMapTypeControlOptions.Style Property

```
property Style: TMapTypeControlStyle
```

Specifies the style of the map type control.

10.150 TMargins Component

Unit: WebUI

Inherits From TBoundingAttribute

The TMargins class represents the margins of a UI element or control. The margins affect how a UI element or control is positioned and sized within the current layout rectangle when using the layout functionality in Elevate Web Builder. Please see the Layout Management topic for more information on how margins are used with the layout functionality.

Properties	Methods	Events

10.151 TMediaControl Component

Unit: WebMedia

Inherits From TBindableColumnControl

The TMediaControl control is the base class for media controls, and contains all of the core media functionality in the form of public methods and protected properties/events that descendant classes can use to create customized media controls.

Properties	Methods	Events
	CanPlayMedia	
	Pause	
	Play	

TMediaControl.CanPlayMedia Method

```
function CanPlayMedia(const MIMEType: String): TCanPlayMedia
```

Call this method to determine if the media control can play a specific type of media.

The MIMEType parameter indicates the MIME type of the media that you wish to test for playback capabilities.

TMediaControl.Pause Method

```
procedure Pause
```

Call this method to pause playback of the media.

TMediaControl.Play Method

```
procedure Play
```

Call this method to start or resume playback of the media.

10.152 TMediaElement Component

Unit: WebUI

Inherits From TElement

The TMediaElement class is the element class for media UI elements, and contains all of the base media playback functionality that is used by the TAudioElement and TVideoElement.

> **Note**
> This element does not provide support for media playback at design-time, and the applicable playback methods and properties are all stubs. Also, this element is never instantiated directly. Only the TAudioElement and TVideoElement UI elements are instantiated.

Properties	Methods	Events
AutoPlay	CanPlayMedia	
CurrentSource	Pause	
CurrentTime	Play	
DefaultPlaybackRate		
Duration		
Ended		
Loop		
Muted		
NetworkState		
Paused		
PlaybackRate		
Preload		
ReadyState		
Seeking		
ShowControls		
Source		
Volume		

TMediaElement.AutoPlay Property

```
property AutoPlay: Boolean
```

Specifies that the media should begin playing as soon as enough data has been loaded to allow playback. The default value is False.

TMediaElement.CurrentSource Property

```
property CurrentSource: String
```

Indicates the URL of the current media being loaded and/or played.

TMediaElement.CurrentTime Property

```
property CurrentTime: Double
```

Indicates the current playback time, in seconds. Setting this property to a new value will cause the media to skip to the specified time.

TMediaElement.DefaultPlaybackRate Property

```
property DefaultPlaybackRate: Double
```

Specifies the default playback rate, with 1 being normal playback, less than 1 being slower playback, and greater than 1 being faster playback. The default value is 1.

Note
The volume will normally be automatically muted when playing media faster or slower than the normal playback rate.

TMediaElement.Duration Property

```
property Duration: Double
```

Indicates the length of the media in seconds. If the duration has not been determined, this property will return 0.

TMediaElement.Ended Property

```
property Ended: Boolean
```

Indicates that the end of the media has been reached.

TMediaElement.Loop Property

```
property Loop: Boolean
```

Specifies that the media playback should automatically restart at the beginning once the end has been reached. The default value is False.

TMediaElement.Muted Property

```
property Muted: Boolean
```

Specifies that the playback volume should be muted. The default valuse is False.

TMediaElement.NetworkState Property

```
property NetworkState: TMediaNetworkState
```

Indicates the network state of the media loading/playback.

TMediaElement.Paused Property

```
property Paused: Boolean
```

Indicates that media playback is paused, either by the user pausing the media via the user interface when the ShowControls property is True, or by the application calling the Pause method. The default value is False.

TMediaElement.PlaybackRate Property

```
property PlaybackRate: Double
```

Specifies the playback rate, with 1 being normal playback, less than 1 being slower playback, and greater than 1 being faster playback. The default value is 1.

Note
The volume will normally be automatically muted when playing media faster or slower than the normal playback rate.

TMediaElement.Preload Property

```
property Preload: TMediaPreload
```

Specifies how much of the current media data should be loaded before playback begins.

TMediaElement.ReadyState Property

```
property ReadyState: TMediaReadyState
```

Indicates whether the media is ready for playback, and if so, a general description of what media data has been loaded.

TMediaElement.Seeking Property

```
property Seeking: Boolean
```

Indicates that media is switching to a new playback location, either by the user changing the playback location in the media via the user interface when the ShowControls property is True, or by the application setting the CurrentTime property.

TMediaElement.ShowControls Property

```
property ShowControls: Boolean
```

Specifies whether the element should show the native user interface for the media being played. The TAudioElement UI element instances will show an audio player interface, and the TVideoElement UI element instances will show a video player interface.

TMediaElement.Source Property

```
property Source: String
```

Specifies the URL of the media to be loaded into the media element. Whenever this property is changed, the existing media is cleared and the new media will start downloading from the web server. Please review the events available for this element in order to get more information on detecting and handling the loading/playback of the media.

TMediaElement.Volume Property

```
property Volume: Integer
```

Specifies the playback volume of the audio for the media. The volume can be set between 0 and 100.

TMediaElement.CanPlayMedia Method

```
function CanPlayMedia(const MIMEType: String): TCanPlayMedia
```

Call this method to determine if the media element can play a specific type of media. The MIMEType parameter indicates the MIME type of the media that you wish to test for playback capabilities.

TMediaElement.Pause Method

```
procedure Pause
```

Call this method to pause playback of the media.

TMediaElement.Play Method

```
procedure Play
```

Call this method to start or resume playback of the media.

10.153 TMenu Component

Unit: WebMenus

Inherits From TMenuControl

The TMenu component represents a vertical menu control that can be used for side menus or popup menus.

Properties	Methods	Events
Background	NewItem	OnAnimationComplete
Border	NewSeparatorItem	OnAnimationsComplete
Corners	Popup	OnEnter
Cursor		OnExit
Hint		OnHide
Opacity		OnItemClick
OutsetShadow		OnMove
TabOrder		OnShow
TabStop		OnSize

TMenu.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TMenu.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TMenu.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TMenu.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TMenu.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TMenu.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TMenu.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the control.

TMenu.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TMenu.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TMenu.NewItem Method

```
function NewItem: TMenuItem
```

Use this method to create a new TMenuItem instance and append it to the current TMenu instance as the last menu item.

TMenu.NewSeparatorItem Method

```
function NewSeparatorItem: TMenuSeparatorItem
```

Use this method to create a new TMenuSeparatorItem instance and append it to the current TMenu instance as the last menu separator item.

TMenu.Popup Method

```
procedure Popup(X,Y: Integer)
```

Use this method to display the menu as a popup menu. When a menu is displayed as a popup, then it will behave as a popup and automatically be hidden whenever the menu loses focus.

TMenu.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TMenu.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TMenu.OnEnter Event

```
property OnEnter: TNotifyEvent
```

This event is triggered when the control obtains input focus.

TMenu.OnExit Event

```
property OnExit: TNotifyEvent
```

This event is triggered when the control loses input focus.

TMenu.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TMenu.OnItemClick Event

```
property OnItemClick: TClickEvent
```

This event is triggered whenever a menu item is clicked.

TMenu.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TMenu.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TMenu.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

10.154 TMenuBar Component

Unit: WebMenus

Inherits From TMenuControl

The TMenu component represents a horizontal menu control that can be used for main menus.

Properties	Methods	Events
Background	NewItem	OnAnimationComplete
Border	NewSeparatorItem	OnAnimationsComplete
Corners		OnEnter
Cursor		OnExit
Hint		OnHide
Opacity		OnItemClick
OutsetShadow		OnMove
TabOrder		OnShow
TabStop		OnSize

TMenuBar.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TMenuBar.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TMenuBar.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TMenuBar.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TMenuBar.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TMenuBar.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TMenuBar.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the control.

TMenuBar.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TMenuBar.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TMenuBar.NewItem Method

```
function NewItem: TMenuBarItem
```

Use this method to create a new TMenuBarItem instance and append it to the current TMenuBar instance as the last menu bar item.

TMenuBar.NewSeparatorItem Method

```
function NewSeparatorItem: TMenuBarSeparatorItem
```

Use this method to create a new TMenuBarSeparatorItem instance and append it to the current TMenuBar instance as the last menu bar separator item.

TMenuBar.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TMenuBar.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TMenuBar.OnEnter Event

```
property OnEnter: TNotifyEvent
```

This event is triggered when the control obtains focus.

TMenuBar.OnExit Event

```
property OnExit: TNotifyEvent
```

This event is triggered when the control loses focus.

TMenuBar.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TMenuBar.OnItemClick Event

```
property OnItemClick: TClickEvent
```

This event is triggered whenever a menu bar item is clicked.

TMenuBar.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TMenuBar.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TMenuBar.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

10.155 TMenuBarItem Component

Unit: WebMenus

Inherits From TMenuItemControl

The TMenuBarItem component represents a textual menu item within a TMenuBar control.

Properties	Methods	Events
AutoWidth		OnClick
Caption		OnEnter
Cursor		OnExit
Enabled		OnHide
Font		OnMouseDown
Hint		OnMouseEnter
Icon		OnMouseLeave
SubMenu		OnMouseMove
		OnMouseUp
		OnShow
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TMenuBarItem.AutoWidth Property

```
property AutoWidth: Boolean
```

Specifies whether the width of the menu bar item should be automatically set based upon the Caption, Icon, and Font properties.

TMenuBarItem.Caption Property

```
property Caption: String
```

Specifies the textual caption to display in the control. The default value is ''.

TMenuBarItem.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TMenuBarItem.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TMenuBarItem.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TMenuBarItem.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TMenuBarItem.Icon Property

```
property Icon: TIconProperties
```

Specifies the properties of the icon used with the control.

TMenuBarItem.SubMenu Property

```
property SubMenu: TMenu
```

Specifies the sub-menu for this menu bar item. Setting this property to an instance of the TMenu control allows the menu bar item to popup the sub-menu when the menu bar item is clicked.

TMenuBarItem.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TMenuBarItem.OnEnter Event

```
property OnEnter: TNotifyEvent
```

This event is triggered when the control obtains focus.

TMenuBarItem.OnExit Event

```
property OnExit: TNotifyEvent
```

This event is triggered when the control loses focus.

TMenuBarItem.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TMenuBarItem.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TMenuBarItem.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TMenuBarItem.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TMenuBarItem.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TMenuBarItem.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TMenuBarItem.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TMenuBarItem.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TMenuBarItem.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TMenuBarItem.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TMenuBarItem.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.156 TMenuBarSeparatorItem Component

Unit: WebMenus

Inherits From TMenuItemControl

The TMenuBarSeparatorItem component represents a menu item separator within a TMenuBar control.

Properties	Methods	Events
		OnHide
		OnShow

TMenuBarSeparatorItem.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TMenuBarSeparatorItem.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

10.157 TMenuControl Component

Unit: WebMenus

Inherits From TControl

The TMenuControl control is the base class for vertical menu controls, and contains all of the core vertical menu functionality in the form of public methods and protected properties/events that descendant classes can use to create customized menu controls.

Properties	Methods	Events
ItemCount	FirstItem	
ItemIndex	LastItem	
Items	MakeItemVisible	
VisibleItemCount	NextItem	
VisibleItems	PriorItem	

TMenuControl.ItemCount Property

```
property ItemCount: Integer
```

Indicates the number of menu items in the menu control.

TMenuControl.ItemIndex Property

```
property ItemIndex: Integer
```

Indicates the index of the currently-selected menu item in the menu control, or -1 if no menu item is selected.

TMenuControl.Items Property

```
property Items[AIndex: Integer]: TMenuItemControl
```

Accesses the menu items in the menu control by index.

TMenuControl.VisibleItemCount Property

```
property VisibleItemCount: Integer
```

Indicates the number of visible menu items in the menu control.

TMenuControl.VisibleItems Property

```
property VisibleItems[AIndex: Integer]: TMenuItemControl
```

Accesses the visible menu items in the menu control by index.

TMenuControl.FirstItem Method

```
procedure FirstItem
```

Use this method to move focus to the first menu item in the menu control.

TMenuControl.LastItem Method

```
procedure LastItem
```

Use this method to move focus to the last menu item in the menu control.

TMenuControl.MakeItemVisible Method

```
procedure MakeItemVisible(AItem: TMenuItemControl)
```

Use this method to ensure that the specified menu item is visible.

TMenuControl.NextItem Method

```
procedure NextItem
```

Use this method to move focus to the next menu item, if one exists, in the menu control.

TMenuControl.PriorItem Method

```
procedure PriorItem
```

Use this method to move focus to the prior menu item, if one exists, in the menu control.

10.158 TMenuItem Component

Unit: WebMenus

Inherits From TMenuItemControl

The TMenuItem component represents a textual menu item within a TMenu control.

Properties	Methods	Events
AutoHeight		OnClick
Caption		OnEnter
Cursor		OnExit
Enabled		OnHide
Font		OnMouseDown
Hint		OnMouseEnter
Icon		OnMouseLeave
SubMenu		OnMouseMove
		OnMouseUp
		OnShow
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TMenuItem.AutoHeight Property

```
property AutoHeight: Boolean
```

Specifies whether the height of the menu item should be automatically set based upon the Caption and Font properties.

TMenuItem.Caption Property

```
property Caption: String
```

Specifies the textual caption to display in the control. The default value is ''.

TMenuItem.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TMenuItem.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TMenuItem.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TMenuItem.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TMenuItem.Icon Property

```
property Icon: TIconProperties
```

Specifies the properties of the icon used with the control.

TMenuItem.SubMenu Property

```
property SubMenu: TMenu
```

Specifies the sub-menu for this menu item. Setting this property to an instance of the TMenu control allows the menu item to popup the sub-menu when the menu item is clicked.

TMenuItem.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TMenuItem.OnEnter Event

```
property OnEnter: TNotifyEvent
```

This event is triggered when the control obtains input focus.

TMenuItem.OnExit Event

```
property OnExit: TNotifyEvent
```

This event is triggered when the control loses input focus.

TMenuItem.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TMenuItem.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TMenuItem.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TMenuItem.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TMenuItem.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TMenuItem.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TMenuItem.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TMenuItem.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TMenuItem.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TMenuItem.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TMenuItem.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.159 TMenuItemControl Component

Unit: WebMenus

Inherits From TControl

The TMenuItemControl control is the base class for menu control items, including normal textual menu items and menu item separators, and contains all of the core vertical menu item functionality in the form of public methods and protected properties/events that descendant classes can use to create customized menu item controls.

Properties	Methods	Events
Index	HideSubMenu	
ParentMenu	ShowSubMenu	

TMenuItemControl.Index Property

```
property Index: Integer
```

Indicates the position of the menu item in the parent menu.

TMenuItemControl.ParentMenu Property

```
property ParentMenu: TMenuControl
```

Indicates the parent menu that contains the menu item.

TMenuItemControl.HideSubMenu Method

```
procedure HideSubMenu
```

Use this method to hide the sub-menu, if one is set for the menu item and is currently visible.

TMenuItemControl.ShowSubMenu Method

```
procedure ShowSubMenu
```

Use this method to show the sub-menu, if one is set for the menu item and is currently not visible.

10.160 TMenuSeparatorItem Component

Unit: WebMenus

Inherits From TMenuItemControl

The TMenuSeparatorItem component represents a menu item separator within a TMenu control.

Properties	Methods	Events
		OnHide
		OnShow

TMenuSeparatorItem.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TMenuSeparatorItem.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

10.161 TMessageDialog Component

Unit: WebForms

Inherits From TDialogControl

The TMessageDialog component represents a message dialog control. Please see the Showing Message Dialogs for more information on using message dialogs.

Properties	Methods	Events
AllowClose	AddButton	OnAnimationComplete
AllowMove		OnAnimationsComplete
ButtonCount		OnClick
Buttons		OnClose
Caption		OnCloseQuery
CloseOnEscape		OnDblClick
Corners		OnHide
Cursor		OnKeyDown
DialogType		OnKeyPress
Message		OnKeyUp
Opacity		OnMouseDown
OutsetShadow		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMouseWheel
		OnMove
		OnResult
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TMessageDialog.AllowClose Property

```
property AllowClose: Boolean
```

Specifies whether the close button should be shown in the caption bar of the dialog.

TMessageDialog.AllowMove Property

```
property AllowMove: Boolean
```

Specifies whether the user can press and hold a mouse or touch on the caption bar and drag the container dialog to a new position.

TMessageDialog.ButtonCount Property

```
property ButtonCount: Integer
```

Indicates the number of dialog buttons added to the dialog.

TMessageDialog.Buttons Property

```
property Buttons[Index: Integer]: TDialogButton
```

Accesses the dialog buttons that have been added to the dialog.

TMessageDialog.Caption Property

```
property Caption: String
```

Specifies the caption to display in the caption bar of the dialog.

TMessageDialog.CloseOnEscape Property

```
property CloseOnEscape: Boolean
```

Specifies whether the dialog is automatically closed when the user hits the Escape key. The default value is True.

Note
If there is a TDialogButton instance on the dialog with its ModalCancel property set to True, then the button will be clicked when the escape key is pressed and this property will ignored.

TMessageDialog.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TMessageDialog.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TMessageDialog.DialogType Property

```
property DialogType: TMsgDlgType
```

Specifies the type of dialog. The dialog type determines which icon is displayed in the dialog.

TMessageDialog.Message Property

```
property Message: String
```

Specifies the message to display in the dialog.

TMessageDialog.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TMessageDialog.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the control.

TMessageDialog.AddButton Method

```
function AddButton(ButtonType: TMsgDlgBtn): TDialogButton
```

Use this method to add a new dialog button to the dialog. The button type determines the caption and behavior of the dialog button.

TMessageDialog.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TMessageDialog.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TMessageDialog.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TMessageDialog.OnClose Event

```
property OnClose: TNotifyEvent
```

This event is triggered when the dialog is closed by the user via the caption bar close button, or when the Close method is called.

TMessageDialog.OnCloseQuery Event

```
property OnCloseQuery: TCloseQueryEvent
```

This event is triggered when the dialog is closed by the user via the caption bar close button, or when the Close method is called.

Return True to allow the close to continue, or False to prevent the dialog from closing.

TMessageDialog.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TMessageDialog.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TMessageDialog.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when a child control has input focus and the user presses a key or key combination.

TMessageDialog.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when a child control has input focus and presses/releases a key or key combination.

TMessageDialog.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when a child control has input focus and the user releases a key or key combination.

TMessageDialog.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TMessageDialog.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TMessageDialog.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TMessageDialog.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TMessageDialog.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TMessageDialog.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

This event is triggered whenever the mouse wheel is rotated forward or backward.

TMessageDialog.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TMessageDialog.OnResult Event

```
property OnResult: TMsgDlgResultEvent
```

This event is triggered after a modal dialog has been closed and a modal result is available.

TMessageDialog.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TMessageDialog.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TMessageDialog.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TMessageDialog.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TMessageDialog.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TMessageDialog.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.162 TModalOverlay Component

Unit: WebForms

Inherits From TControl

The TModalOverlay component represents the modal overlay area that is used to cover all existing controls when a form is shown modally. This component provides access to the overlay so that its appearance can be customized. Please see the Creating and Showing Forms topic for more information on showing forms.

Properties	Methods	Events
Background		
CloseOnClick		

TModalOverlay.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TModalOverlay.CloseOnClick Property

```
property CloseOnClick: Boolean
```

Specifies whether clicking on the modal overlay will automatically close all visible modal forms. The default value is False.

10.163 TMultiLineEdit Component

Unit: WebEdits

Inherits From TMultiLineEditControl

The TMultiLineEdit component represents a multi-line edit control. A a multi-line edit control allows the user to directly enter an input value using the keyboard, and the input value can contain multiple lines and be word-wrapped.

Properties	Methods	Events
Alignment		OnAnimationComplete
Cursor		OnAnimationsComplete
DataColumn		OnChange
DataSet		OnClick
Direction		OnDblClick
Enabled		OnEnter
Font		OnExit
Hint		OnHide
Lines		OnKeyDown
MaxLength		OnKeyPress
ReadOnly		OnKeyUp
ScrollBars		OnMouseDown
ScrollSupport		OnMouseEnter
SpellCheck		OnMouseLeave
TabOrder		OnMouseMove
TabStop		OnMouseUp
Text		OnMouseWheel
WordWrap		OnMove
		OnScroll
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchScroll
		OnTouchStart

TMultiLineEdit.Alignment Property

```
property Alignment: TContentAlignment
```

Specifies the alignment of the input value for the control.

TMultiLineEdit.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TMultiLineEdit.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TMultiLineEdit.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TMultiLineEdit.Direction Property

```
property Direction: TContentDirection
```

Specifies the direction in which the text is displayed/edited.

TMultiLineEdit.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TMultiLineEdit.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TMultiLineEdit.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TMultiLineEdit.Lines Property

```
property Lines: TStrings
```

Specifies the lines to display and edit in the control.

Note
Updating this property will automatically update the Text property, and vice-versa.

TMultiLineEdit.MaxLength Property

```
property MaxLength: Integer
```

Specifies the maximum allowable length, in characters, of the Text property for the control. A value of 0 specifies an unlimited allowable length.

TMultiLineEdit.ReadOnly Property

```
property ReadOnly: Boolean
```

Specifies whether the control's input value can be modified by the user. The default value is False.

Note
The input value can always be programmatically modified.

TMultiLineEdit.ScrollBars Property

```
property ScrollBars: TScrollBars
```

Specifies which scrollbars to show, if any.

Note
Even if this property is set to sbHorizontal, sbVertical, or sbBoth, a scrollbar will only be shown if the size of the contents of the control exceed the client rectangle for the control.

TMultiLineEdit.ScrollSupport Property

```
property ScrollSupport: TScrollSupport
```

Specifies the directions in which the control can be scrolled, if any.

Note
This property only applies to scrolling via touch and mouse wheel movements, and does not apply to programmatic scrolling or scrolling via the scroll bars themselves.

TMultiLineEdit.SpellCheck Property

```
property SpellCheck: Boolean
```

Specifies whether spell-checking will be enabled for the control.

TMultiLineEdit.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TMultiLineEdit.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TMultiLineEdit.Text Property

```
property Text: String
```

Specifies the control's input value as a string.

Note
Updating this property will automatically update the Lines property, and vice-versa.

TMultiLineEdit.WordWrap Property

```
property WordWrap: Boolean
```

Specifies whether the content should be word-wrapped.

TMultiLineEdit.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TMultiLineEdit.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TMultiLineEdit.OnChange Event

```
property OnChange: TNotifyEvent
```

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TMultiLineEdit.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TMultiLineEdit.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TMultiLineEdit.OnEnter Event

```
property OnEnter: TNotifyEvent
```

This event is triggered when the control obtains input focus.

TMultiLineEdit.OnExit Event

```
property OnExit: TNotifyEvent
```

This event is triggered when the control loses input focus.

TMultiLineEdit.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TMultiLineEdit.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when the control has input focus and the user presses a key or key combination.

TMultiLineEdit.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TMultiLineEdit.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when the control has input focus and the user releases a key or key combination.

TMultiLineEdit.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TMultiLineEdit.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TMultiLineEdit.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TMultiLineEdit.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TMultiLineEdit.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TMultiLineEdit.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

This event is triggered whenever the mouse wheel is rotated forward or backward.

TMultiLineEdit.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TMultiLineEdit.OnScroll Event

```
property OnScroll: TNotifyEvent
```

This event is triggered whenever a scrollable control is scrolled horizontally or vertically.

TMultiLineEdit.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TMultiLineEdit.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TMultiLineEdit.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TMultiLineEdit.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TMultiLineEdit.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TMultiLineEdit.OnTouchScroll Event

```
property OnTouchScroll: TTouchScrollEvent
```

This event is triggered whenever a touch moves in any direction over a touch-scroll-enabled control.

TMultiLineEdit.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.164 TMultiLineEditControl Component

Unit: WebEdits

Inherits From TEditControl

The TMultiLineEditControl control is the base class for multi-line edit controls, and contains all of the core multi-line edit functionality in the form of public methods and protected properties/events that descendant classes can use to create customized multi-line edit controls.

Properties	Methods	Events

10.165 TObjectElement Component

Unit: WebUI

Inherits From TWebElement

The TObjectElement class is the element class for browser objects (plugins), and contains all of the browser object functionality in the form of public methods and properties/events that control classes can use to create browser object controls.

> **Note**
> This element does not provide support for objects at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events
MIMEType		
Params		

TObjectElement.MIMEType Property

```
property MIMEType: String
```

Specifies the MIME type of the object resource that will be loaded when the URL property is specified. This information is used by the browser to determine which browser plugin to load in order to allow interaction with the resource.

An event handler can be attached to the OnLoad event to execute code when the object is loaded.

TObjectElement.Params Property

```
property Params: TStrings
```

Specifies any parameters for the plugin that will be executed by the browser when the object resource specified by the URL property is loaded. The parameters are specified as key/value pairs:

```
Parameter=Value
```

Note
Please see the help for the applicable browser plugin in order to find out which parameters are supported by the plugin.

10.166 TObjectList Component

Unit: WebCore

Inherits From TAbstractList

The TObjectList class is used to manage a list of class instances (objects). The constructor for the class accepts a single Boolean parameter that indicates whether the class will retain ownership of all managed objects and destroy them automatically when the TObjectList class instance is destroyed.

Properties	Methods	Events
Count	Add	
Objects	AddObjects	
OwnsObjects	Clear	
Sorted	Create	
	Delete	
	Dequeue	
	Exchange	
	Find	
	First	
	IndexOf	
	Insert	
	Last	
	Move	
	Next	
	Pop	
	Prior	
	Push	
	Queue	
	Remove	
	Requeue	
	Sort	

TObjectList.Count Property

```
property Count: Integer
```

Indicates the number of objects managed by the class.

TObjectList.Objects Property

```
property Objects[Index: Integer]: TObject
```

Allows indexed access to all objects managed by the class. If the class owns its managed objects, then all managed objects are automatically destroyed when the class is destroyed.

TObjectList.OwnsObjects Property

```
property OwnsObjects: Boolean
```

Indicates whether the class owns its managed objects and will automatically destroy all managed objects when the class is destroyed.

TObjectList.Sorted Property

```
property Sorted: Boolean
```

Specifies that the list of objects is sorted. When the list of objects is sorted, any operations that modify the list automatically trigger a re-sort of the objects in the list.

Note
This property is only useful for TObjectList descendant classes that override two protected object comparison methods that are called in order to compare the names of objects, as well as the objects themselves.

TObjectList.Add Method

```
function Add(AObject: TObject): Integer
```

Use this method to add an object to the list of managed objects. The object will be added to the end of the list, and the index of the object in the list will be returned.

TObjectList.AddObjects Method

```
procedure AddObjects(AList: TObjectList)
```

Use this method to add all of the objects from a source list to the list of managed objects in this class.

TObjectList.Clear Method

```
procedure Clear
```

Use this method to clear all object instances from the list. If the list's OwnsObjects property is True, then all instances will automatically be freed as well.

TObjectList.Create Method

```
constructor Create

constructor Create(AOwnsObjects: Boolean)
```

Use this method to create a new instance of the TObjectList class. This method is overloaded, and the second version of the method contains an optional AOwnsObjects parameter that indicates whether the object list instance will own its contained object instances and automatically free them when the object list itself is freed.

TObjectList.Delete Method

```
procedure Delete(Index: Integer; FreeOwnedObject: Boolean=True)
```

Use this method to remove an object from the list of managed objects by its index. The FreeOwnedObject parameter will cause the object instance to be destroyed if the list's OwnsObjects property is True.

TObjectList.Dequeue Method

```
function Dequeue: TObject
```

Use this method to obtain a reference to the first object (index 0) in the list of managed objects and remove the object from the list.

TObjectList.Exchange Method

```
procedure Exchange(Source: Integer; Dest: Integer)
```

Use this method to exchange the positions of two managed objects in the list.

TObjectList.Find Method

```
function Find(const Value: String; NearestMatch: Boolean=False):
      Integer
```

Use this method to perform a binary search of the list of objects. The Sorted property must be True or calling this method will result in an exception being raised.

Note
This method is only useful for TObjectList descendant classes that override two protected object comparison methods that are called in order to compare the names of objects, as well as the objects themselves.

TObjectList.First Method

```
function First: TObject
```

Use this method to return the first object (index 0) in the list of managed objects.

TObjectList.IndexOf Method

```
function IndexOf(AObject: TObject): Integer
```

Use this method to return the index of a particular object in the list of managed objects.

TObjectList.Insert Method

```
procedure Insert(Index: Integer; AObject: TObject)
```

Use this method to insert an object at a specific index in the list of managed objects.

TObjectList.Last Method

```
function Last: TObject
```

Use this method to return the last object (index Count-1) in the list of managed objects.

TObjectList.Move Method

```
procedure Move(Source: Integer; Dest: Integer)
```

Use this method to move the object specified by the Source index to the position specified by the Dest index.

Note
It is important to remember that a move operation is equivalent to a delete of the object at the Source index followed by an insert of the object at the Dest index. This means that you must account for the fact that the Dest index may need to be decremented if the Source index is less than the Dest index.

TObjectList.Next Method

```
function Next(AObject: TObject; Wrap: Boolean=False): TObject
```

Use this method to return the next object, relative to the object passed as the first parameter, in the list of managed objects. The Wrap parameter determines if the method should wrap around to the start of the list if the passed object is the last object in the list.

TObjectList.Pop Method

```
function Pop: TObject
```

Use this method to obtain a reference to the last object (index Count-1) in the list of managed objects and remove the object from the list.

TObjectList.Prior Method

```
function Prior(AObject: TObject; Wrap: Boolean=False): TObject
```

Use this method to return the prior object, relative to the object passed as the first parameter, in the list of managed objects. The Wrap parameter determines if the method should wrap around to the end of the list if the passed object is the first object in the list.

TObjectList.Push Method

```
procedure Push(AObject: TObject)
```

Use this method to add an object to the end of the list of managed objects.

TObjectList.Queue Method

```
procedure Queue(AObject: TObject)
```

Use this method to add an object to the end of the list of managed objects.

TObjectList.Remove Method

```
function Remove(AObject: TObject; FreeOwnedObject:
      Boolean=True): Integer
```

Use this method to remove the specified object from the list of managed objects. The FreeOwnedObject parameter will cause the object instance to be destroyed if the list's OwnsObjects property is True.

TObjectList.Requeue Method

```
procedure Requeue(AObject: TObject)
```

Use this method to insert an object at the beginning of the list of managed objects.

TObjectList.Sort Method

```
procedure Sort
```

Use this method to sort the managed objects.

Note
This method is only useful for TObjectList descendant classes that override two protected object comparison methods that are called in order to compare the names of objects, as well as the objects themselves.

10.167 TOutsetShadow Component

Unit: WebUI

Inherits From TShadow

The TOutsetShadow class represents the outset shadow of a UI element or control. The outset shadow appears behind the bounds of the element.

Properties	Methods	Events

10.168 TOverviewMapControlOptions Component

Unit: WebMaps

Inherits From TMapOption

The TOverviewMapControlOptions class controls how the overview map control is configured in a TMap control. These overview map control options correspond to the overview map control options available for maps in the Google Maps API.

Properties	Methods	Events
Opened		

TOverviewMapControlOptions.Opened Property

```
property Opened: Boolean
```

Specifies whether the overview map control is opened.

10.169 TPadding Component

Unit: WebUI

Inherits From TBoundingAttribute

The TPadding class represents the padding within the client area of a UI element or control. The padding affects the size of the client rectangle for a UI element or control: larger padding values decrease the size of the client rectangle, while smaller padding values increase the size of the client rectangle.

Properties	Methods	Events

10.170 TPage Component

Unit: WebPages

Inherits From TControl

The TPage component represents a page within a TPagePanel control. A page is a nested container, and can be dynamically added and removed from a page panel control. Each page instance contains a reference to a tab control via its Tab property. The properties in the tab can be modified to affect the tab caption and how the tab is sized and formatted.

Properties	Methods	Events
Background	Close	OnAnimationComplete
Border	SetActive	OnAnimationsComplete
Corners		OnClick
Cursor		OnClose
Index		OnCloseQuery
InsetShadow		OnDblClick
Padding		OnHide
ParentPagePanel		OnMouseDown
Tab		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TPage.Background Property

```
property Background: TBackground
```

Specifies the background for the control.

TPage.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TPage.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TPage.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TPage.Index Property

```
property Index: Integer
```

Specifies the index of the page in its parent page panel's pages.

TPage.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Specifies the inset shadow for the control.

TPage.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

TPage.ParentPagePanel Property

```
property ParentPagePanel: TPagePanelControl
```

Indicates the parent page panel that contains the page.

TPage.Tab Property

```
property Tab: TTab
```

Specifies the properties of the tab for the page.

TPage.Close Method

```
procedure Close
```

Use this method to close the page. When this method is called, the OnCloseQuery event is triggered, followed by the OnClose event. If the OnCloseQuery event handler returns True, then the page will be hidden before the OnClose event handler is triggered. After the OnClose event handler is executed, the page will be removed from its parent page panel and disposed of.

TPage.SetActive Method

```
procedure SetActive
```

Use this method to make the current page the active page in the parent page panel control.

TPage.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TPage.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TPage.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TPage.OnClose Event

```
property OnClose: TNotifyEvent
```

This event is triggered when the page is closed by the user via the tab close button, or when the Close method is called.

TPage.OnCloseQuery Event

```
property OnCloseQuery: TCloseQueryEvent
```

This event is triggered when the page is closed by the user via the tab close button, or when the Close method is called.

Return True to allow the close to continue, or False to prevent the page from closing.

TPage.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TPage.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TPage.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TPage.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TPage.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TPage.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TPage.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TPage.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TPage.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TPage.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TPage.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TPage.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TPage.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TPage.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.171 TPagePanel Component

Unit: WebPages

Inherits From TPagePanelControl

The TPagePanel component represents a page panel control. A page panel control contains 0 or more instances of TPage controls that can be dynamically added and removed from the page panel control. Each page instance contains a reference to a tab control via its Tab property. The properties in the tab can be modified to affect the tab caption and how the tab is sized and formatted.

Properties	Methods	Events
Border		OnAnimationComplete
Corners		OnAnimationsComplete
Cursor		OnHide
Gutter		OnMove
Opacity		OnPageChange
Padding		OnPageChanged
PageNavigation		OnShow
TabOrder		OnSize
TabStop		
TabsVisible		

TPagePanel.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TPagePanel.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TPagePanel.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TPagePanel.Gutter Property

```
property Gutter: Integer
```

Specifies the size, in pixels, of the blank space to show to the left of the tabs for the page panel control.

TPagePanel.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TPagePanel.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

TPagePanel.PageNavigation Property

```
property PageNavigation: Boolean
```

Specifies whether the pages of the page panel control can be navigated using an interactive navigation bar when the TabsVisible property is False.

TPagePanel.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TPagePanel.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TPagePanel.TabsVisible Property

```
property TabsVisible: Boolean
```

Specifies whether the tabs for the pages should be shown. The default value is True.

Note
When this property is False, the PageNavigation property can be used to show or hide an interactive navigation bar.

TPagePanel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TPagePanel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TPagePanel.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TPagePanel.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TPagePanel.OnPageChange Event

```
property OnPageChange: TPageChangeEvent
```

This event is triggered whenever the ActivePage changes. To prevent the page change, return False from any event handler attached to this event.

TPagePanel.OnPageChanged Event

```
property OnPageChanged: TPageChangeEvent
```

TPagePanel.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TPagePanel.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

10.172 TPagePanelControl Component

Unit: WebPages

Inherits From TControl

The TPagePanelControl control is the base class for page panel controls, and contains all of the core page panel functionality in the form of public methods and protected properties/events that descendant classes can use to create customized page panel controls.

Properties	Methods	Events
ActivePage	FirstPage	
PageCount	LastPage	
Pages	MakePageVisible	
	NavigateToPage	
	NewPage	
	NextPage	
	PriorPage	
	RemoveActivePage	
	ScrollNext	
	ScrollPrior	

TPagePanelControl.ActivePage Property

```
property ActivePage: TPage
```

Specifies the active page in the page panel control.

TPagePanelControl.PageCount Property

```
property PageCount: Integer
```

Indicates the number of pages in the page panel control.

TPagePanelControl.Pages Property

```
property Pages[AIndex: Integer]: TPage
```

Accesses the pages in the page panel control by index.

TPagePanelControl.FirstPage Method

```
function FirstPage: TPage
```

Use this method to make the first page in the control the active page.

TPagePanelControl.LastPage Method

```
function LastPage: TPage
```

Use this method to make the last page in the control the active page.

TPagePanelControl.MakePageVisible Method

```
procedure MakePageVisible(APage: TPage)
```

Use this method to ensure that the specified page's tab is visible.

TPagePanelControl.NavigateToPage Method

```
function NavigateToPage(APage: TPage): TPage
```

Use this method to make the specified page the active page in the control.

TPagePanelControl.NewPage Method

```
function NewPage: TPage
```

Use this method to create a new page. The new page will be positioned after all other existing pages.

TPagePanelControl.NextPage Method

```
function NextPage: TPage
```

Use this method to make the next page, relative to the specified page, the active page in the control.

TPagePanelControl.PriorPage Method

```
function PriorPage: TPage
```

Use this method to make the prior page, relative to the specified page, the active page in the control.

TPagePanelControl.RemoveActivePage Method

```
procedure RemoveActivePage
```

Use this method to remove the active page, if one exists. If the ActivePage property is nil, then this method does nothing.

TPagePanelControl.ScrollNext Method

```
procedure ScrollNext
```

Use this method to scroll the page tabs to the right by one tab.

TPagePanelControl.ScrollPrior Method

```
procedure ScrollPrior
```

Use this method to scroll the page tabs to the left by one tab.

10.173 TPaint Component

Unit: WebPaint

Inherits From TControl

The TPaint component represents a drawing/painting control. A paint control can be used for general drawing, charting, animation, and more. It contains a reference to a TCanvasElement instance that can be used to perform all drawing operations, and automatically resizes the canvas instance to match the dimensions of the control.

Properties	Methods	Events
Canvas		OnAnimationComplete
Cursor		OnClick
Hint		OnDblClick
Opacity		OnHide
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TPaint.Canvas Property

```
property Canvas: TCanvasElement
```

This property provdes access to a TCanvasElement instance that can be used to perform all drawing operations within the dimensions of the control.

TPaint.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TPaint.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TPaint.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TPaint.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TPaint.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TPaint.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TPaint.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TPaint.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TPaint.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TPaint.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TPaint.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TPaint.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TPaint.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TPaint.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TPaint.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TPaint.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TPaint.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TPaint.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TPaint.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.174 TPanControlOptions Component

Unit: WebMaps

Inherits From TMapOption

The TPanControlOptions class controls how the pan control is configured in a TMap control. These pan control options correspond to the pan control options available for maps in the Google Maps API.

Properties	Methods	Events
Position		

TPanControlOptions.Position Property

```
property Position: TMapControlPosition
```

Specifies the position of the pan control.

10.175 TPanel Component

Unit: WebCtnrs

Inherits From TPanelControl

The TPanel component represents a panel control with a border and a caption bar with minimize/restore and close buttons.

Properties	Methods	Events
ActivateOnClick		OnAnimationComplete
AutoSize		OnAnimationsComplete
Background		OnCaptionBarDblClick
Border		OnClick
CaptionBar		OnClose
Client		OnCloseQuery
Corners		OnDblClick
Cursor		OnHide
Hint		OnKeyDown
Opacity		OnKeyPress
OutsetShadow		OnKeyUp
ScrollBars		OnMinimize
ScrollSupport		OnMouseDown
TabOrder		OnMouseEnter
TabStop		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMouseWheel
		OnMove
		OnRestore
		OnScroll
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchScroll
		OnTouchStart

TPanel.ActivateOnClick Property

```
property ActivateOnClick: Boolean
```

Specifies whether the control should automatically be brought to the front when it, or any child controls, are clicked.

TPanel.AutoSize Property

```
property AutoSize: TAutoSize
```

Specifies how (if at all) the control should automatically be sized based upon the child controls placed in the panel.

TPanel.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TPanel.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TPanel.CaptionBar Property

```
property CaptionBar: TPanelCaptionBar
```

Specifies the properties of the caption bar for the control.

TPanel.Client Property

```
property Client: TPanelClient
```

Specifies the properties of the client area for the control.

TPanel.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TPanel.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TPanel.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TPanel.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TPanel.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the control.

TPanel.ScrollBars Property

```
property ScrollBars: TScrollBars
```

Specifies which scrollbars to show, if any.

Note
Even if this property is set to sbHorizontal, sbVertical, or sbBoth, a scrollbar will only be shown if the size of the contents and/or the child controls of the control exceed the client rectangle for the control.

TPanel.ScrollSupport Property

```
property ScrollSupport: TScrollSupport
```

Specifies the directions in which the control can be scrolled, if any.

Note
This property only applies to scrolling via touch and mouse wheel movements, and does not apply to programmatic scrolling or scrolling via the scroll bars themselves.

TPanel.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TPanel.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TPanel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TPanel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TPanel.OnCaptionBarDblClick Event

```
property OnCaptionBarDblClick: TNotifyEvent
```

This event is triggered when the caption bar is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TPanel.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TPanel.OnClose Event

```
property OnClose: TNotifyEvent
```

This event is triggered when the panel is closed by the user via the caption bar close button, or when the Close method is called.

TPanel.OnCloseQuery Event

```
property OnCloseQuery: TCloseQueryEvent
```

This event is triggered when the panel is closed by the user via the caption bar close button, or when the Close method is called. Return True to allow the close to continue, or False to prevent the panel from closing.

TPanel.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TPanel.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TPanel.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when a child control has input focus and the user presses a key or key combination.

TPanel.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when a child control has input focus and presses/releases a key or key combination.

TPanel.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when a child control has input focus and the user releases a key or key combination.

TPanel.OnMinimize Event

```
property OnMinimize: TNotifyEvent
```

This event is triggered when the panel is minimized.

TPanel.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TPanel.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TPanel.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TPanel.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TPanel.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TPanel.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

This event is triggered whenever the mouse wheel is rotated forward or backward.

TPanel.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TPanel.OnRestore Event

```
property OnRestore: TNotifyEvent
```

This event is triggered when the panel is restored from a minimized state.

TPanel.OnScroll Event

```
property OnScroll: TNotifyEvent
```

This event is triggered whenever a scrollable control is scrolled horizontally or vertically.

TPanel.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TPanel.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TPanel.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TPanel.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TPanel.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TPanel.OnTouchScroll Event

```
property OnTouchScroll: TTouchScrollEvent
```

This event is triggered whenever a touch moves in any direction over a touch-scroll-enabled control.

TPanel.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.176 TPanelCaptionBar Component

Unit: WebCtnrs

Inherits From TCaptionBarControl

The TPanelCaptionBar component represents the caption bar for a TPanel control, and includes a caption and minimize/restore and close buttons.

Properties	Methods	Events
Alignment		
AllowClose		
AllowMinimize		
AllowMove		
Background		
Caption		
Cursor		
Font		
Icon		
Padding		

TPanelCaptionBar.Alignment Property

```
property Alignment: TContentAlignment
```

Specifies the alignment of the caption in the caption bar.

TPanelCaptionBar.AllowClose Property

```
property AllowClose: Boolean
```

Specifies whether the close button should be shown in the caption bar.

TPanelCaptionBar.AllowMinimize Property

```
property AllowMinimize: Boolean
```

Specifies whether the minimize/restore button should be shown in the caption bar.

TPanelCaptionBar.AllowMove Property

```
property AllowMove: Boolean
```

Specifies whether the user can press and hold a mouse or touch on the caption bar and drag the container panel to a new position.

TPanelCaptionBar.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TPanelCaptionBar.Caption Property

```
property Caption: String
```

Specifies the caption to display in the caption bar.

TPanelCaptionBar.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TPanelCaptionBar.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TPanelCaptionBar.Icon Property

```
property Icon: TIconProperties
```

Specifies the properties of the icon used with the caption bar.

TPanelCaptionBar.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

10.177 TPanelClient Component

Unit: WebCtnrs

Inherits From TComponent

The TPanelClient component represents the client area for a TPanel control.

Properties	Methods	Events
Background		
InsetShadow		
Padding		

TPanelClient.Background Property

```
property Background: TBackground
```

Specifies the background for the control.

TPanelClient.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Specifies the inset shadow for the control.

TPanelClient.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

10.178 TPanelControl Component

Unit: WebCtnrs

Inherits From TScrollableControl

The TPanelControl control is the base class for panel controls, and contains all of the core panel functionality in the form of public methods and protected properties/events that descendant classes can use to create customized panel controls.

Properties	Methods	Events
	Close	

TPanelControl.Close Method

```
procedure Close
```

Use this method to close the panel. When this method is called, the OnCloseQuery event is triggered, followed by the OnClose event. If the OnCloseQuery event handler returns True, then the panel will be hidden before the OnClose event is triggered.

10.179 TParser Component

Unit: WebCore

Inherits From TObject

The TParser class is a parser class for parsing JSON strings, and is used for loading forms, control interfaces, and datasets (columns, rows, and transaction operations). It can be used as a general-purpose JSON parser in your applications.

Properties	Methods	Events
PropertyNameType	CheckString	
Token	CheckSymbol	
TokenLiteral	CheckToken	
TokenString	ErrorIfNotSkipString	
	ErrorIfNotSkipSymbol	
	ErrorIfNotSkipToken	
	ErrorIfNotString	
	ErrorIfNotSymbol	
	ErrorIfNotToken	
	ExpectedError	
	GetBoolean	
	GetFloat	
	GetInteger	
	GetString	
	Initialize	
	NextToken	
	SkipArray	
	SkipObject	
	SkipProperty	
	SkipPropertyValue	
	SkipString	
	SkipSymbol	
	SkipToken	

TParser.PropertyNameType Property

```
property PropertyNameType: Char
```

Specifies the Token type to use when parsing JSON object property names. The default value is tkString, which means that the parser will expect property names to be specified in the same way as strings: enclosed in double-quote (") characters.

TParser.Token Property

```
property Token: Char
```

Indicates the current token character or token type. The following special token types are used for non-character tokens:

Token Type	Description
tkTerm	Indicates that the current token is the terminating character (#0)
tkSymbol	Indicates that the current token is a symbol
tkString	Indicates that the current token is a string enclosed in double-quote (") characters
tkInteger	Indicates that the current token is an integer
tkFloat	Indicates that the current token is a floating-point number

TParser.TokenLiteral Property

```
property TokenLiteral: String
```

Indicates the current token in its literal form. For strings, this means that the double quote (") characters are included in the value returned by this property.

TParser.TokenString Property

```
property TokenString: String
```

Indicates the current token. For strings, this means that the double quote (") characters are **not** included in the value returned by this property.

TParser.CheckString Method

```
function CheckString(const Value: String): Boolean
```

Use this method to determine if the current token is a string.

TParser.CheckSymbol Method

```
function CheckSymbol(const Value: String): Boolean
```

Use this method to determine if the current token is a symbol.

TParser.CheckToken Method

```
function CheckToken(Value: Char): Boolean
```

Use this method to determine if the current token is a particular character or type of token.

TParser.ErrorIfNotSkipString Method

```
procedure ErrorIfNotSkipString(const Value: String)
```

Use this method to determine if the current token is a string and, if so, to move to the next token. If the current token is not a string, then an exception will be raised.

TParser.ErrorIfNotSkipSymbol Method

```
procedure ErrorIfNotSkipSymbol(const Value: String)
```

Use this method to determine if the current token is a symbol and, if so, to move to the next token. If the current token is not a symbol, then an exception will be raised.

TParser.ErrorIfNotSkipToken Method

```
procedure ErrorIfNotSkipToken(Value: Char)
```

Use this method to determine if the current token is a particular character or type of token and, if so, to move to the next token. If the current token is not the particular character or type of token, then an exception will be raised.

TParser.ErrorIfNotString Method

```
procedure ErrorIfNotString(const Value: String)
```

Use this method to determine if the current token is a string. If the current token is not a string, then an exception will be raised.

TParser.ErrorIfNotSymbol Method

```
procedure ErrorIfNotSymbol(const Value: String)
```

Use this method to determine if the current token is a symbol. If the current token is not a symbol, then an exception will be raised.

TParser.ErrorIfNotToken Method

```
procedure ErrorIfNotToken(Value: Char)
```

Use this method to determine if the current token is a particular character or type of token. If the current token is not the particular character or type of token, then an exception will be raised.

TParser.ExpectedError Method

```
procedure ExpectedError(const Value: String)
```

Use this method to raise an exception due to a parsing error, such as a case when a certain token literal was expected and a different literal was found instead.

TParser.GetBoolean Method

```
function GetBoolean: Boolean
```

Use this method to retrieve the current token as a boolean value.

Note
If the Token property is not equal to tkSymbol and the TokenString property is not a valid JSON boolean literal (true, false), an exception will be raised.

TParser.GetFloat Method

```
function GetFloat: Double
```

Use this method to retrieve the current token as a float value.

Note
If the Token property is not equal to tkFloat or tkInteger, an exception will be raised.

TParser.GetInteger Method

```
function GetInteger: Integer
```

Use this method to retrieve the current token as an integer value.

Note
If the Token property is not equal to tkInteger, an exception will be raised.

TParser.GetString Method

```
function GetString: String
```

Use this method to retrieve the current token as a string value, without any enclosing double-quote (") characters.

Note
If the Token property is not equal to tkString, an exception will be raised.

TParser.Initialize Method

```
procedure Initialize(const TextToParse: String; FullNumbers:
      Boolean=True; APropertyNameType: Char=tkString)
```

Use this method to initialize the parser with a string containing the text that is to be parsed. This method will automatically "seed" the parser by calling the NextToken method.

TParser.NextToken Method

```
procedure NextToken
```

Use this method to move to the next token.

TParser.SkipArray Method

```
function SkipArray: Boolean
```

Use this method to skip over a JSON array. If the current token is not the left bracket ([) character, then this method will return False.

TParser.SkipObject Method

```
function SkipObject: Boolean
```

Use this method to skip over a JSON object. If the current token is not the left brace ({) character, then this method will return False.

TParser.SkipProperty Method

```
procedure SkipProperty
```

Use this method to skip over an entire JSON property, including the property name and value. The SkipPropertyValue method is used to skip over the value.

TParser.SkipPropertyValue Method

```
procedure SkipPropertyValue
```

Use this method to skip over a JSON property value. If the JSON property value is an object, then the SkipObject method is used to skip over the value. If the JSON property value is an array, then the SkipArray method is used to skip over the value. Otherwise, the NextToken method is used to skip over the value.

TParser.SkipString Method

```
function SkipString(const Value: String): Boolean
```

Use this method to determine if the current token is a string and, if so, to move to the next token.

TParser.SkipSymbol Method

```
function SkipSymbol(const Value: String): Boolean
```

Use this method to determine if the current token is a symbol and, if so, to move to the next token.

TParser.SkipToken Method

```
function SkipToken(Value: Char): Boolean
```

Use this method to determine if the current token is a particular character or type of token and, if so, to move to the next token.

10.180 TPasswordEdit Component

Unit: WebEdits

Inherits From TEditControl

The TPasswordEdit component represents a password edit control. An edit control allows the user to directly enter an input value using the keyboard but, instead of showing the input value in the edit control, the input characters are masked so that they cannot be seen.

Properties	Methods	Events
Alignment		OnAnimationComplete
Cursor		OnAnimationsComplete
DataColumn		OnChange
DataSet		OnClick
Direction		OnDblClick
Enabled		OnEnter
Font		OnExit
Hint		OnHide
MaxLength		OnKeyDown
ReadOnly		OnKeyPress
TabOrder		OnKeyUp
TabStop		OnMouseDown
Text		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TPasswordEdit.Alignment Property

```
property Alignment: TContentAlignment
```

Specifies the alignment of the input value for the control.

TPasswordEdit.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TPasswordEdit.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TPasswordEdit.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TPasswordEdit.Direction Property

```
property Direction: TContentDirection
```

Specifies the direction in which the text is displayed/edited.

TPasswordEdit.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TPasswordEdit.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TPasswordEdit.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TPasswordEdit.MaxLength Property

```
property MaxLength: Integer
```

Specifies the maximum allowable length, in characters, of the Text property for the control. A value of 0 specifies an unlimited allowable length.

TPasswordEdit.ReadOnly Property

```
property ReadOnly: Boolean
```

Specifies whether the control's input value can be modified by the user. The default value is False.

Note
The input value can always be programmatically modified.

TPasswordEdit.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TPasswordEdit.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TPasswordEdit.Text Property

```
property Text: String
```

Specifies the control's input value as a string.

TPasswordEdit.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TPasswordEdit.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TPasswordEdit.OnChange Event

```
property OnChange: TNotifyEvent
```

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TPasswordEdit.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TPasswordEdit.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TPasswordEdit.OnEnter Event

```
property OnEnter: TNotifyEvent
```

This event is triggered when the control obtains input focus.

TPasswordEdit.OnExit Event

```
property OnExit: TNotifyEvent
```

This event is triggered when the control loses input focus.

TPasswordEdit.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TPasswordEdit.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when the control has input focus and the user presses a key or key combination.

TPasswordEdit.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TPasswordEdit.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when the control has input focus and the user releases a key or key combination.

TPasswordEdit.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TPasswordEdit.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TPasswordEdit.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TPasswordEdit.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TPasswordEdit.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TPasswordEdit.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TPasswordEdit.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TPasswordEdit.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TPasswordEdit.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TPasswordEdit.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TPasswordEdit.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TPasswordEdit.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.181 TPasswordInputElement Component

Unit: WebUI

Inherits From TInputElement

The TPasswordInputElement class is the base element class for password input elements, and contains all of the password input functionality in the form of public methods and properties/events that control classes can use to create password input controls.

> **Note**
> This element does not provide support for password input elements at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events

10.182 TPersistent Component

Unit: WebCore

Inherits From TObject

The TPersistent class is the base class for classes that require the ability to save/load their published properties to/from JSON strings. All of the functionality in the TPersistent class is protected and normally only accessible to descendant classes. The TWriter class is used by TPersistent descendants for outputting properties to a JSON string, and the TReader class for parsing properties from a JSON string.

All TComponent, TCollection, TCollectionItem, and TControl classes are TPersistent-descendant classes, and all contain functionality for automatically loading their published properties. In addition, the TFormControl class contains special functionality for loading the published properties for a form instance and all of its contained components/controls. This functionality is used to load the published properties for a form class when an instance of the class is created.

Properties	Methods	Events

10.183 TPersistentStorage Component

Unit: WebComps

Inherits From TObject

The TPersistentStorage object encapsulates the HTML5 local storage functionality, which allows the application to store name-value pairs of strings using the host web browser's storage facilities. This type of storage is preferable to cookies (TCookies) due to the fact that cookies are normally limited to around 4k of storage.

Note
The component library includes two global instances of this class called LocalStorage and SessionStorage in the WebComps unit that should be used instead of creating new instances of the class. The LocalStorage instance represents the persistent local storage in the host web browser that is available across browser instances/sessions, whereas the SessionStorage instance represents the session-only storage in the host web browser that is cleared whenever the browser is closed.

Properties	Methods	Events
Count	Clear	OnChange
Items	ClearAll	
Keys	Exists	
	Set	

TPersistentStorage.Count Property

```
property Count: Integer
```

Indicates the number of items defined.

TPersistentStorage.Items Property

```
property Items[Index: Integer]: String

property Items[const AName: String]: String
```

Allows access to all defined items, either by index or by name.

TPersistentStorage.Keys Property

```
property Keys[Index: Integer]: String
```

Allows access to all defined item names by index.

TPersistentStorage.Clear Method

```
procedure Clear(const AName: String)
```

Use this method to clear the specified item.

Note
Use the Exists method to determine if a item exists before trying to clear the item.

TPersistentStorage.ClearAll Method

```
procedure ClearAll
```

Use this method to clear all items.

TPersistentStorage.Exists Method

```
function Exists(const AName: String): Boolean
```

Use this method to determine if the specified item exists.

TPersistentStorage.Set Method

```
procedure Set(const AName: String; const Value: String)
```

Use this method to set an item value. The Name parameter is the item name and the Value parameter is the item value.

TPersistentStorage.OnChange Event

```
property OnChange: TStorageChangeEvent
```

This event is triggered whenever the contents of the persistent local storage is changed by a different session in the browser.

Note
This event is only triggered for the persistent local storage and not the session-only local storage.

10.184 TPlugin Component

Unit: WebBrwsr

Inherits From TWebControl

The TPlugin component represents a web browser plugin container control that can interact with any type of object resource that is supported by a registered plugin in the web browser, including MP3/MP4 audio files, PDF document files, and MPEG video files.

Properties	Methods	Events
Background		OnAnimationComplete
Border		OnAnimationsComplete
Corners		OnHide
Cursor		OnLoad
DataColumn		OnMove
DataSet		OnShow
Hint		OnSize
InsetShadow		OnUnload
Loaded		
MimeType		
Opacity		
OutsetShadow		
Padding		
Params		
URL		

TPlugin.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TPlugin.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TPlugin.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TPlugin.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TPlugin.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TPlugin.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TPlugin.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TPlugin.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Specifies the inset shadow for the control.

TPlugin.Loaded Property

```
property Loaded: Boolean
```

Indicates whether the object resource specified by the URL property has been loaded.

An event handler can be attached to the OnLoad event to execute code when the object resource is loaded.

TPlugin.MimeType Property

```
property MimeType: String
```

Specifies the MIME type of the object resource that will be loaded when the URL property is specified. This information is used by the browser to determine which browser plugin to load in order to allow interaction with the resource.

An event handler can be attached to the OnLoad event to execute code when the object is loaded.

TPlugin.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TPlugin.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the control.

TPlugin.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

TPlugin.Params Property

```
property Params: TStrings
```

Specifies any parameters for the plugin that will be executed by the browser when the object resource specified by the URL property is loaded. The parameters are specified as key/value pairs:

```
Parameter=Value
```

Note
Please see the help for the applicable browser plugin in order to find out which parameters are supported by the plugin.

TPlugin.URL Property

```
property URL: String
```

Specifies the URL for the object resource. Whenever the URL property changes, the OnUnload event is triggered immediately. The OnLoad event is triggered once the object resource has been loaded.

TPlugin.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TPlugin.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TPlugin.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TPlugin.OnLoad Event

```
property OnLoad: TNotifyEvent
```

This event is triggered when the file specified by the URL property has been completely loaded by a browser plugin.

TPlugin.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TPlugin.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TPlugin.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TPlugin.OnUnload Event

```
property OnUnload: TNotifyEvent
```

This event is triggered when the currently-loaded file specified by the URL property has been unloaded.

10.185 TPoint Component

Unit: WebUI

Inherits From TObject

The TPoint class represents the X/Y integer coordinates of a point. It is used with the TElement class and descendant classes for calculating coordinates during event management for an element, or elements.

Properties	Methods	Events
X	Assign	
Y	Clear	
	Create	
	Equals	
	Offset	

TPoint.X Property

```
property X: Integer
```

Specifies the horizontal position of the point.

TPoint.Y Property

```
property Y: Integer
```

Specifies the vertical position of the point.

TPoint.Assign Method

```
procedure Assign(APoint: TPoint)

procedure Assign(AX,AY: Integer)
```

Use this method to assign a source point to the point instance.

TPoint.Clear Method

```
procedure Clear
```

Use this method to set both of the point coordinates to 0.

TPoint.Create Method

```
constructor Create(AX,AY: Integer)
```

Use this method to create a new instance of the TPoint class using the provided X and Y coordinates.

TPoint.Equals Method

```
function Equals(APoint: TPoint): Boolean
```

Use this method to test if two points have the same coordinates.

TPoint.Offset Method

```
procedure Offset(AX,AY: Integer)
```

Use this method to offset the point. Offsetting a point increments or decrements its X and Y properties using the AX and AY parameters, respectively.

10.186 TProgressBar Component

Unit: WebProgs

Inherits From TControl

The TProgressBar component represents a progress bar control for showing visual progress between a minimum and maximum set of values.

Properties	Methods	Events
Background		OnAnimationComplete
Border		OnAnimationsComplete
Corners		OnClick
Cursor		OnDblClick
Indicator		OnHide
MaxValue		OnMouseDown
MinValue		OnMouseEnter
Padding		OnMouseLeave
Position		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TProgressBar.Background Property

```
property Background: TBackground
```

Specifies the background for the control.

TProgressBar.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TProgressBar.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TProgressBar.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TProgressBar.Indicator Property

```
property Indicator: TProgressBarIndicator
```

Accesses the properties of the progress bar indicator.

TProgressBar.MaxValue Property

```
property MaxValue: Integer
```

Specifies the maximum progress bar value. The default value is 100.

TProgressBar.MinValue Property

```
property MinValue: Integer
```

Specifies the minimum progress bar value. The default value is 0.

TProgressBar.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

TProgressBar.Position Property

```
property Position: Integer
```

Specifies the current progress bar position between the MinValue and MaxValue properties.

TProgressBar.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TProgressBar.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TProgressBar.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TProgressBar.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TProgressBar.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TProgressBar.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TProgressBar.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TProgressBar.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TProgressBar.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TProgressBar.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TProgressBar.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TProgressBar.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TProgressBar.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TProgressBar.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TProgressBar.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TProgressBar.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TProgressBar.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.187 TProgressBarIndicator Component

Unit: WebProgs

Inherits From TComponent

The TProgressBarIndicator component represents the indicator portion of a TProgressBar control.

Properties	Methods	Events
Background		
Border		
Corners		

TProgressBarIndicator.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TProgressBarIndicator.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TProgressBarIndicator.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

10.188 TProgressDialog Component

Unit: WebForms

Inherits From TDialogControl

The TProgressDialog component represents a progress dialog control. Please see the Showing Progress Dialogs topic for more information on using progress dialogs.

Properties	Methods	Events
Corners		OnAnimationComplete
Cursor		OnAnimationsComplete
Message		OnClick
Opacity		OnClose
OutsetShadow		OnDblClick
		OnHide
		OnKeyDown
		OnKeyPress
		OnKeyUp
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMouseWheel
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TProgressDialog.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TProgressDialog.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TProgressDialog.Message Property

```
property Message: String
```

Specifies the message to display in the dialog.

TProgressDialog.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TProgressDialog.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the control.

TProgressDialog.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TProgressDialog.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TProgressDialog.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TProgressDialog.OnClose Event

```
property OnClose: TNotifyEvent
```

This event is triggered when the dialog's Close method is called.

TProgressDialog.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TProgressDialog.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TProgressDialog.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when a child control has input focus and the user presses a key or key combination.

TProgressDialog.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when a child control has input focus and presses/releases a key or key combination.

TProgressDialog.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when a child control has input focus and the user releases a key or key combination.

TProgressDialog.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TProgressDialog.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TProgressDialog.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TProgressDialog.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TProgressDialog.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TProgressDialog.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

This event is triggered whenever the mouse wheel is rotated forward or backward.

TProgressDialog.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TProgressDialog.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TProgressDialog.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TProgressDialog.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TProgressDialog.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TProgressDialog.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TProgressDialog.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.189 TRadioButton Component

Unit: WebBtns

Inherits From TStateButtonControl

The TRadioButton component represents a radio button control. A radio button control is a state control that is used in groups to allow the user to select a specific control by using a mouse click, or by pushing the spacebar or enter key. A set of radio button controls are considered in the same group when they share the same parent container. You can change the position of the radio button controls within a container by modifying their LayoutOrder property.

Properties	Methods	Events
AutoWidth		OnAnimationComplete
Caption		OnAnimationsComplete
Cursor		OnChange
DataColumn		OnClick
DataSet		OnEnter
Enabled		OnExit
Font		OnHide
Hint		OnKeyDown
ReadOnly		OnKeyPress
SelectionState		OnKeyUp
TabOrder		OnMouseDown
TabStop		OnMouseEnter
ValueSelected		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TRadioButton.AutoWidth Property

```
property AutoWidth: Boolean
```

Specifies whether the width of the radio button should be automatically set based upon the Caption and Font properties.

TRadioButton.Caption Property

```
property Caption: String
```

Specifies the caption for the control.

Note
The caption area of a control is also clickable with the mouse or touch interface.

TRadioButton.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TRadioButton.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TRadioButton.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TRadioButton.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the control is enabled or disabled. When a control is disabled, it cannot obtain input focus and is displayed in a disabled state. The default value is True.

TRadioButton.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TRadioButton.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TRadioButton.ReadOnly Property

```
property ReadOnly: Boolean
```

Specifies whether the control's input value can be modified by the user. The default value is False.

Note

The input value can always be programmatically modified.

TRadioButton.SelectionState Property

```
property SelectionState: TSelectionState
```

Specifies the selection state of the control.

Note
The ssIndeterminate selection state can only be set programmatically. When the user toggles the selection for the control, it will alternate between the ssSelected and ssUnselected selection states.

TRadioButton.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TRadioButton.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TRadioButton.ValueSelected Property

```
property ValueSelected: String
```

Specifies the textual value to use for the selected state when reading and writing data to and from the data column that the control is bound to. The default value is the string representation of the layout order of the control within its parent container control.

TRadioButton.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TRadioButton.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TRadioButton.OnChange Event

```
property OnChange: TNotifyEvent
```

This event is triggered whenever the input value of the control is changed, either by the user or programmatically.

TRadioButton.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TRadioButton.OnEnter Event

```
property OnEnter: TNotifyEvent
```

This event is triggered when the control obtains input focus.

TRadioButton.OnExit Event

```
property OnExit: TNotifyEvent
```

This event is triggered when the control loses input focus.

TRadioButton.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TRadioButton.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when the control has input focus and the user presses a key or key combination.

TRadioButton.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when the control has input focus and the user presses/releases a key or key combination.

TRadioButton.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when the control has input focus and the user releases a key or key combination.

TRadioButton.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TRadioButton.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TRadioButton.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TRadioButton.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TRadioButton.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TRadioButton.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TRadioButton.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TRadioButton.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TRadioButton.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TRadioButton.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TRadioButton.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TRadioButton.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.190 TReader Component

Unit: WebCore

Inherits From TObject

The TReader class is a class used by TPersistent-descendant classes to load their published properties from JSON strings, and is used for loading forms and control interfaces. It can be used as a general-purpose JSON reader in your applications.

When a TReader instance is created, the constructor allows you to specify the date-time format to use when reading date-time properties.

Properties	Methods	Events
GlobalComponent	BeginArray	
Level	BeginObject	
RootComponent	Create	
	EndArray	
	EndObject	
	EndOfArray	
	EndOfObject	
	GetPropertyName	
	Initialize	
	IsArray	
	IsBoolean	
	IsNull	
	IsObject	
	IsString	
	MoreArrayElements	
	MoreProperties	
	ReadBoolean	
	ReadDateTime	
	ReadFloat	
	ReadInteger	
	ReadString	
	SkipArrayElement	
	SkipProperty	
	SkipPropertyName	
	SkipPropertySeparator	
	SkipPropertyValue	

TReader.GlobalComponent Property

```
property GlobalComponent: TComponent
```

Specifies an optional global component to use with the reader. This property is used with forms and databases to determine how to apply any component reference fixups that need to occur after the form or database is loaded. The default value is nil.

Note

If this property is not specified, then the Owner property of the RootComponent will be used as the instance to use for applying component reference fixups.

TReader.Level Property

```
property Level: Integer
```

Indicates the current nesting level for any JSON objects and/or arrays. Whenever the TReader class begins to read an object or array using the BeginObject or BeginArray methods, the nesting level is incremented. Whenever the EndObject or EndArray methods are called, the nesting level is decremented.

TReader.RootComponent Property

```
property RootComponent: TComponent
```

Specifies an optional root component to use with the reader. The root component is the component that is supposed to be the owner of all TPersistent instances being read from the incoming JSON string. This property is used with form controls to specify the owner of all component instances created during the loading of the form. The default value is nil.

TReader.BeginArray Method

```
procedure BeginArray
```

Use this method to begin reading an array. If the current token in the incoming JSON string is not a left bracket ([), an exception will be raised.

TReader.BeginObject Method

```
procedure BeginObject
```

Use this method to begin reading an object. If the current token in the incoming JSON string is not a left brace ({), an exception will be raised.

TReader.Create Method

```
constructor Create(ADateTimeFormat: TDateTimeFormat=dtfRaw)
```

Use this method to create a new instance of the TReader class. The optional ADateTimeFormat parameter indicates whether date and time values should be handled as an ISO 8601 date and time string value, or as a raw Unix date and time integer value (the number of milliseconds since midnight, January 1, 1970).

TReader.EndArray Method

```
procedure EndArray
```

Use this method to end reading an array. If the current token in the incoming JSON string is not a right bracket (]), an exception will be raised.

TReader.EndObject Method

```
procedure EndObject
```

Use this method to end reading an object. If the current token in the incoming JSON string is not a right brace (}), an exception will be raised.

TReader.EndOfArray Method

```
function EndOfArray: Boolean
```

Use this method to determine if the current token in the incoming JSON string is a right bracket (]).

TReader.EndOfObject Method

```
function EndOfObject: Boolean
```

Use this method to determine if the current token in the incoming JSON string is a right brace (}).

TReader.GetPropertyName Method

```
function GetPropertyName: String
```

Use this method to read a property name, without any enclosing double-quote (") characters (if applicable).

TReader.Initialize Method

```
procedure Initialize(const Value: String; ACompressedProperties:
      Boolean=False)
```

Use this method to initialize the reader with a JSON string to read.

TReader.IsArray Method

```
function IsArray: Boolean
```

Use this method to determine if the current token in the incoming JSON string is a left bracket ([).

TReader.IsBoolean Method

```
function IsBoolean: Boolean
```

Use this method to determine if the current token in the incoming JSON string is a boolean value (true or false).

TReader.IsNull Method

```
function IsNull: Boolean
```

Use this method to determine if the current token in the incoming JSON string is a null literal.

TReader.IsObject Method

```
function IsObject: Boolean
```

Use this method to determine if the current token in the incoming JSON string is a left brace ({).

TReader.IsString Method

```
function IsString: Boolean
```

Use this method to determine if the current token in the incoming JSON string is a string value ("").

TReader.MoreArrayElements Method

```
function MoreArrayElements: Boolean
```

Use this method to determine if the current token in the incoming JSON string is a comma (,).

TReader.MoreProperties Method

```
function MoreProperties: Boolean
```

Use this method to determine if the current token in the incoming JSON string is a comma (,).

TReader.ReadBoolean Method

```
function ReadBoolean: Boolean
```

Use this method to read a boolean value. If the current token in the incoming JSON string is not a valid JSON boolean literal (true, false), an exception will be raised.

TReader.ReadDateTime Method

```
function ReadDateTime: DateTime
```

Use this method to read a date-time value. How a date-time value is read is controlled by the TDateTimeFormat parameter in the TReader class constructor.

TReader.ReadFloat Method

```
function ReadFloat: Double
```

Use this method to read a float value. If the current token in the incoming JSON string is not a valid JSON float or integer literal, an exception will be raised.

TReader.ReadInteger Method

```
function ReadInteger: Integer
```

Use this method to read an integer value. If the current token in the incoming JSON string is not a valid JSON integer literal, an exception will be raised.

TReader.ReadString Method

```
function ReadString: String
```

Use this method to read a string value, without any enclosing double-quote (") characters. If the current token in the incoming JSON string is not a valid JSON string literal, an exception will be raised.

TReader.SkipArrayElement Method

```
procedure SkipArrayElement
```

Use this method to skip over a JSON array element. The TPersistent class uses this method to skip an array element when the class instance does not know how to properly load the array element.

TReader.SkipProperty Method

```
procedure SkipProperty
```

Use this method to skip over a JSON object property. The TPersistent class uses this method to skip a property when the class instance does not know how to properly load the property.

TReader.SkipPropertyName Method

```
procedure SkipPropertyName
```

Use this method to skip over a JSON object property name. The TPersistent class uses this method to skip past the property name after reading the name and determining that the property exists in the class for the instance being loaded.

TReader.SkipPropertySeparator Method

```
procedure SkipPropertySeparator
```

Use this method to skip over a JSON object property separator (:). The TPersistent class uses this method to skip past the property name/separator after reading the property name and determining that the property exists in the class for the instance being loaded.

TReader.SkipPropertyValue Method

```
procedure SkipPropertyValue
```

Use this method to skip over a JSON object property value. The TPersistent class uses this method to skip past any property values that it cannot load for any reason.

10.191 TRect Component

Unit: WebUI

Inherits From TObject

The TRect class represents a rectangle using integer coordinates. It is used with the TElement class and descendant classes for calculating rectangle coordinates for all of the element's dimensional functionality, such as layout management.

Properties	Methods	Events
Bottom	Anchor	
Empty	Assign	
Height	Clear	
Left	Contains	
Right	Create	
Top	Equals	
Width	Fit	
	Inflate	
	Interpolate	
	Normalize	
	Offset	
	Scale	
	Union	

TRect.Bottom Property

```
property Bottom: Integer
```

Specifies the bottom Y coordinate of the rectangle.

TRect.Empty Property

```
property Empty: Boolean
```

Indicates whether the rectangle is empty. A rectangle is considered empty if its width or height is 0.

TRect.Height Property

```
property Height: Integer
```

Indicates the height of the rectangle (read-only).

TRect.Left Property

```
property Left: Integer
```

Specifies the left X coordinate of the rectangle.

TRect.Right Property

```
property Right: Integer
```

Specifies the right X coordinate of the rectangle.

TRect.Top Property

```
property Top: Integer
```

Specifies the top Y coordinate of the rectangle.

TRect.Width Property

```
property Width: Integer
```

Indicates the width of the rectangle (read-only).

TRect.Anchor Method

```
procedure Anchor
```

Use this method to anchor the rectangle. Anchoring a rectangle changes its Left and Top properties to 0, and adjusts the Right and Bottom properties accordingly so that the rectangle retains its same Width and Height.

TRect.Assign Method

```
procedure Assign(ARect: TRect)

procedure Assign(ALeft,ATop,ARight,ABottom: Integer)
```

Use this method to assign a source rectangle, or source rectangle coordinates, to the rectangle instance.

TRect.Clear Method

```
procedure Clear
```

Use this method to set all of the rectangle coordinates to 0.

TRect.Contains Method

```
function Contains(X,Y: Integer): Boolean
```

TRect.Create Method

```
constructor Create(ALeft,ATop,ARight,ABottom: Integer=0)
```

Use this method to create a new instance of the TRect class. The optional ALeft, ATop, ARight, and ABottom parameters will initialize the instance with the provided coordinates.

TRect.Equals Method

```
function Equals(ARect: TRect): Boolean
```

Use this method to test if two rectangles have the same coordinates.

TRect.Fit Method

```
procedure Fit(AWidth,AHeight: Integer)
```

Use this method to proportionally adjust the width and height of the rectangle so that it fits within the specified AWidth and AHeight parameters.

TRect.Inflate Method

```
procedure Inflate(ALeft,ATop,ARight,ABottom: Integer)
```

Use this method to inflate the rectangle. Inflating a rectangle increments or decrements its Left, Top, Right, and Bottom properties using the specified parameters.

TRect.Interpolate Method

```
procedure Interpolate(ARect: TRect; AAmount: Double)
```

Use this method to calculate a new rectangle using the difference between the coordinates of the rectangle and a source rectangle multiplied by a specified distance amount. This calculation is useful for moving a rectangle along a given linear path between a source rectangle and a destination rectangle.

TRect.Normalize Method

```
procedure Normalize
```

Use this method to normalize the rectangle. Normalizing a rectangle changes (if necessary) its Left, Top, Right, and Bottom properties so that the Left property is less than or equal to the Right property and the Top property is less than or equal to the Bottom property.

TRect.Offset Method

```
procedure Offset(ALeft: Integer; ATop: Integer)
```

Use this method to offset the rectangle. Offsetting a rectangle increments or decrements its Left and Top properties using the ALeft and ATop parameters, respectively.

TRect.Scale Method

```
procedure Scale(ARatio: Double)
```

Use this method to proportionally scale the coordinates of the rectangle using the specified ARatio parameter.

TRect.Union Method

```
procedure Union(ARect: TRect)
```

Use this method to union the coordinates of the rectangle with another rectangle. A union operation is a max operation on each pair of coordinates, taking the smallest of the two rectangles' Left and Top properties, and the largest of the two rectangles' Right and Bottom properties.

10.192 TRepeatControl Component

Unit: WebCtrls

Inherits From TControl

The TRepeatControl control is the base class for controls that have a repeatable click behavior, and contains all of the repeatable click functionality in the form of public methods and protected properties/events that descendant classes can use to create customized controls.

Properties	Methods	Events

10.193 TRotateControlOptions Component

Unit: WebMaps

Inherits From TMapOption

The TRotateControlOptions class controls how the rotate control is configured in a TMap control. These rotate control options correspond to the rotate control options available for maps in the Google Maps API.

Properties	Methods	Events
Position		

TRotateControlOptions.Position Property

```
property Position: TMapControlPosition
```

Specifies the position of the rotate control.

10.194 TScript Component

Unit: WebComps

Inherits From TComponent

The TScript component represents a dynamically-loaded script and is used to introduce external JavaScript into the global execution context from a URL, as opposed to being linked via the emitted HTML with the application.

Properties	Methods	Events
URL		OnError
		OnLoad

TScript.URL Property

```
property URL: String
```

Specifies the full URL for the script to be loaded dynamically. Whenever this property is changed, the existing script is removed from the global execution context and the new script is downloaded from the specified URL. Once the script has been downloaded and loaded, the OnLoad event will be triggered. If there was an error downloading the script, then the OnError event will be triggered.

TScript.OnError Event

```
property OnError: TNotifyEvent
```

This event is triggered when the download of the script encounters an error.

TScript.OnLoad Event

```
property OnLoad: TNotifyEvent
```

This event is triggered when the download of the script is complete and the script is loaded into the application's global execution context.

10.195 TScrollableControl Component

Unit: WebCtrls

Inherits From TControl

The TScrollableControl control is the base class for scrollable controls, and contains all of the scrolling functionality in the form of public methods and protected properties/events that descendant classes can use to create customized scrollable controls.

Properties	Methods	Events

10.196 TScrollPanel Component

Unit: WebCtnrs

Inherits From TScrollPanelControl

The TScrollPanel component represents a scrollable panel control. A scrollable panel control is useful when you need only a portion of a form or other type of container control to be scrollable.

Properties	Methods	Events
ActivateOnClick		OnAnimationComplete
Background		OnAnimationsComplete
Client		OnClick
Cursor		OnDblClick
Hint		OnHide
Opacity		OnKeyDown
OutsetShadow		OnKeyPress
ScrollBars		OnKeyUp
ScrollSupport		OnMouseDown
TabOrder		OnMouseEnter
TabStop		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMouseWheel
		OnMove
		OnScroll
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchScroll
		OnTouchStart

TScrollPanel.ActivateOnClick Property

```
property ActivateOnClick: Boolean
```

Specifies whether the control should automatically be brought to the front when it, or any child controls, are clicked.

TScrollPanel.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TScrollPanel.Client Property

```
property Client: TScrollPanelClient
```

Specifies the properties of the client area for the control.

TScrollPanel.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TScrollPanel.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TScrollPanel.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TScrollPanel.OutsetShadow Property

```
property OutsetShadow: TOutsetShadow
```

Specifies the outset shadow for the control.

TScrollPanel.ScrollBars Property

```
property ScrollBars: TScrollBars
```

Specifies which scrollbars to show, if any.

> **Note**
> Even if this property is set to sbHorizontal, sbVertical, or sbBoth, a scrollbar will only be shown if the size of the contents and/or the child controls of the control exceed the client rectangle for the control.

TScrollPanel.ScrollSupport Property

```
property ScrollSupport: TScrollSupport
```

Specifies the directions in which the control can be scrolled, if any.

> **Note**
> This property only applies to scrolling via touch and mouse wheel movements, and does not apply to programmatic scrolling or scrolling via the scroll bars themselves.

TScrollPanel.TabOrder Property

```
property TabOrder: Integer
```

Specifies the position of the control in the tabbing order for the control's Parent container control. The default value is the last tab position in the container control, or -1 if the Parent property is nil.

TScrollPanel.TabStop Property

```
property TabStop: Boolean
```

Specifies whether the control will participate in the tabbing order within the control's Parent container control. The default value is True.

TScrollPanel.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TScrollPanel.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TScrollPanel.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TScrollPanel.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TScrollPanel.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TScrollPanel.OnKeyDown Event

```
property OnKeyDown: TKeyDownEvent
```

This event is triggered when a child control has input focus and the user presses a key or key combination.

TScrollPanel.OnKeyPress Event

```
property OnKeyPress: TKeyPressEvent
```

This event is triggered when a child control has input focus and presses/releases a key or key combination.

TScrollPanel.OnKeyUp Event

```
property OnKeyUp: TKeyUpEvent
```

This event is triggered when a child control has input focus and the user releases a key or key combination.

TScrollPanel.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TScrollPanel.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TScrollPanel.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TScrollPanel.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TScrollPanel.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TScrollPanel.OnMouseWheel Event

```
property OnMouseWheel: TMouseWheelEvent
```

This event is triggered whenever the mouse wheel is rotated forward or backward.

TScrollPanel.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TScrollPanel.OnScroll Event

```
property OnScroll: TNotifyEvent
```

This event is triggered whenever a scrollable control is scrolled horizontally or vertically.

TScrollPanel.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TScrollPanel.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TScrollPanel.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TScrollPanel.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TScrollPanel.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TScrollPanel.OnTouchScroll Event

```
property OnTouchScroll: TTouchScrollEvent
```

This event is triggered whenever a touch moves in any direction over a touch-scroll-enabled control.

TScrollPanel.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.197 TScrollPanelClient Component

Unit: WebCtnrs

Inherits From TComponent

The TScrollPanelClient component represents the client area for a TScrollPanel control.

Properties	Methods	Events
Background		
InsetShadow		
Padding		

TScrollPanelClient.Background Property

```
property Background: TBackground
```

Specifies the background for the control.

TScrollPanelClient.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Specifies the inset shadow for the control.

TScrollPanelClient.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

10.198 TScrollPanelControl Component

Unit: WebCtnrs

Inherits From TScrollableControl

The TScrollPanelControl control is the base class for scroll panel controls, and contains all of the core scroll panel functionality in the form of public methods and protected properties/events that descendant classes can use to create customized scroll panel controls.

Properties	Methods	Events

10.199 TServerRequest Component

Unit: WebHTTP

Inherits From TComponent

The TServerRequest component represents a dynamic HTTP request to the web server from which the application was loaded or, provided that the web server allows it, a different origin (protocol, host, and port).

Properties	Methods	Events
CrossOriginCredentials	Cancel	OnComplete
Method	Execute	OnProgress
MethodName	ParseXML	OnStart
Params	Reset	
Password		
RequestContent		
RequestHeaders		
RequestQueue		
RequestURL		
ResponseContent		
ResponseContentType		
ResponseHeaders		
StatusCode		
StatusText		
Timeout		
URL		
UserName		

TServerRequest.CrossOriginCredentials Property

```
property CrossOriginCredentials: Boolean
```

Specifies whether HTTP cookies and/or authentication headers are sent with any requests to an origin (protocol, host, and port) that is different than the origin for the application. The default value is False.

TServerRequest.Method Property

```
property Method: TRequestMethod
```

Specifies the HTTP method to use for the web server request.

TServerRequest.MethodName Property

```
property MethodName: String
```

Indicates the descriptive name of the HTTP method specified by the Method property.

TServerRequest.Params Property

```
property Params: TStrings
```

Specifies the URL parameters for the web server request. The Params property is automatically set up to use the equals (=) character for separating the name/value pairs, so you can use the Values property to set them by name. If specifying the parameters directly as name=value strings in the string list, each parameter should be specified as a separate string.

TServerRequest.Password Property

```
property Password: String
```

Specifies the password to use for authenticating the request with the web server, if required by the web server for the URL specified for the request. Both the UserName and the Password property are required in order to properly authenticate the request.

TServerRequest.RequestContent Property

```
property RequestContent: TStrings
```

Specifies any additional content that is being sent with the web server request.

Note
If this property is specified, then please make sure to also specify a **Content-Length** request header in the RequestHeaders property.

TServerRequest.RequestHeaders Property

```
property RequestHeaders: TStrings
```

Specifies any additional headers that are being sent with the web server request.

Note
All necessary headers will be automatically populated by the web browser, so only use this property for specifying headers that are necessary for the web server request, such as a **Content-Type** or **Content-Length** header when sending content to the web server in the RequestContent property.

TServerRequest.RequestQueue Property

```
property RequestQueue: TServerRequestQueue
```

If the current server request instance was created by a TServerRequestQueue component, then this property will contain a reference to the instance of the queue component that created the request.

TServerRequest.RequestURL Property

```
property RequestURL: String
```

Indicates the complete URL that will be used with the server request, including any query parameters specified via the Params property.

TServerRequest.ResponseContent Property

```
property ResponseContent: TStrings
```

Indicates any content that was returned from the web server in the response to the server request.

Note
This property will only be populated once the OnComplete event has been triggered.

TServerRequest.ResponseContentType Property

```
property ResponseContentType: String
```

Indicates the value of the Content-Type HTTP header, if present, that was returned from the web server in the response to the server request. If no Content-Type header was returned, then this property will return an empty string ('').

TServerRequest.ResponseHeaders Property

```
property ResponseHeaders: TStrings
```

Indicates any headers that were returned from the web server in the response to the server request.

Note
This property will only be populated once the OnComplete event has been triggered.

TServerRequest.StatusCode Property

```
property StatusCode: Integer
```

Indicates the status code returned by the web server in response to the server request.

Note
This property will only be populated once the OnComplete event has been triggered.

TServerRequest.StatusText Property

```
property StatusText: String
```

Indicates the status message returned by the web server in response to the server request.

Note
This property will only be populated once the OnComplete event has been triggered.

TServerRequest.Timeout Property

```
property Timeout: Integer
```

Specifies how long the server request should wait, in milliseconds, for a successful connection to the server before returning an error. The default value is 0, which means to wait a browser-defined number of milliseconds.

TServerRequest.URL Property

```
property URL: TServerRequestURL
```

Specifies the URL of the resource that the server request wishes to get data from, or send data to.

Warning
It is highly recommended that you only use relative URLs only in this property. Most modern web browsers will prevent server requests that don't access resources from the same origin (protocol, host, port number) from executing, unless the web server specifically allows such a request. Such a request is referred to as a cross-origin resource sharing request. For more information on how to permit such requests with the Elevate Web Builder Web Server, please see the Configuring the Web Server topic.

TServerRequest.UserName Property

```
property UserName: String
```

Specifies the user name to use for authenticating the request with the web server, if required by the web server for the URL specified for the request. Both the UserName and the Password property are required in order to properly authenticate the request.

TServerRequest.Cancel Method

```
procedure Cancel(KeepExecuting: Boolean=True)
```

Use this method to abort a pending web server request.

TServerRequest.Execute Method

```
procedure Execute
```

Use this method to execute a web server request after specifying the Method and URL properties, at a minimum.

TServerRequest.ParseXML Method

```
function ParseXML: TDocument
```

Use this method to parse XML content returned as a response to the web server request. This method returns a TDocument web browser DOM object instance that can be used to manipulate the various nodes of the XML document as DOM elements.

Please see the WebDOM unit included with the Elevate Web Builder framework for the various interfaces to the DOM objects such as TDocument available in the web browser.

TServerRequest.Reset Method

```
procedure Reset
```

TServerRequest.OnComplete Event

```
property OnComplete: TServerRequestEvent
```

This event is triggered when the server request is complete. Use the StatusCode property to determine the status code returned by the web server and, subsequently, whether the request was successful or not.

TServerRequest.OnProgress Event

```
property OnProgress: TServerRequestProgressEvent
```

TServerRequest.OnStart Event

```
property OnStart: TServerRequestEvent
```

This event is triggered when the server request is started. A server request is started when the Execute method is called.

10.200 TServerRequestQueue Component

Unit: WebHTTP

Inherits From TComponent

The TServerRequestQueue component represents a queue of dynamic HTTP requests to the web server from which the application was loaded or, provided that the web server allows it, a different origin (protocol, host, and port). Serialization allows server requests to be executed in a specific order. If executed individually, such requests would normally be executed asynchronously by the web browser and would not have a predictable completion order.

Properties	Methods	Events
NumPendingRequests	AddRequest	
	CancelAllRequests	
	CancelRequest	
	ExecuteRequests	
	GetNewRequest	
	NextPendingRequest	

TServerRequestQueue.NumPendingRequests Property

```
property NumPendingRequests: Integer
```

Indicates the number of requests currently awaiting execution.

Note

The value returned by this property does **not** include the currently-executing request, if one exists. If a currently-executing request fails for any reason, then the value returned by this property **will** include the failed request when this property is referenced from an OnComplete event handler.

TServerRequestQueue.AddRequest Method

```
procedure AddRequest(Value: TServerRequest)
```

Use this method to add a new request to the end of the queue of web server requests. When using this method, always use the GetNewRequest to obtain a new request that can be modified before calling this method.

TServerRequestQueue.CancelAllRequests Method

```
procedure CancelAllRequests
```

Use this method to cancel all pending web server requests in the queue.

TServerRequestQueue.CancelRequest Method

```
procedure CancelRequest(KeepExecuting: Boolean=True)
```

Use this method to cancel the currently executing web server request in the queue. After the current request is cancelled, the next request in the queue will be executed.

TServerRequestQueue.ExecuteRequests Method

```
procedure ExecuteRequests
```

Use this method to execute any pending web server requests in the queue.

TServerRequestQueue.GetNewRequest Method

```
function GetNewRequest: TServerRequest
```

Use this method to allocate a new web server request to use with the queue. After modifying the request as required, call the AddRequest method to add the request to the queue.

TServerRequestQueue.NextPendingRequest Method

```
function NextPendingRequest: TServerRequest
```

10.201 TSet Component

Unit: WebCore

Inherits From TObject

The TSet class represents a set. It is used with the TElement class and descendant classes for determining which aspects of the element has changed during a batch update, but it can be used for any general purpose.

Properties	Methods	Events
Count	Add	
Max	All	
	Assign	
	Copy	
	Create	
	Empty	
	Except	
	Exists	
	Initialize	
	Intersect	
	IsEmpty	
	Range	
	Remove	
	Union	

TSet.Count Property

```
property Count: Integer
```

Specifies how many items are in the set.

TSet.Max Property

```
property Max: Integer
```

Specifies the maximum number of items that are, or have been, in the set. This property is useful when you need to iterate over the set and test whether particular values are in the set.

TSet.Add Method

```
function Add(Value: Integer): Boolean
```

Use this method to add an item to the set.

TSet.All Method

```
procedure All(ACount: Integer)
```

Use this method to include all items from 0 to ACount-1.

TSet.Assign Method

```
procedure Assign(Value: TSet)
```

Use this method to assign the contents of a source set to the set.

TSet.Copy Method

```
function Copy: TSet
```

Use this method to make a new copy of the set and return it as the result.

TSet.Create Method

```
constructor Create(const AValues: TIntegerArray)
```

Use this method to create a new instance of the TSet class. The AValues parameter is an array of integer values that will be used to initialize the set.

TSet.Empty Method

```
procedure Empty
```

Use this method to remove all items from the set.

TSet.Except Method

```
procedure Except(Value: TSet)
```

Use this method to remove all items from the set that exist in the set passed as the parameter.

TSet.Exists Method

```
function Exists(Value: Integer): Boolean
```

Use this method to determine if an item exists in the set.

TSet.Initialize Method

```
procedure Initialize(const AValues: TIntegerArray)
```

Use this method to initialize a set using an array of integer values representing the items that should be in the set.

TSet.Intersect Method

```
procedure Intersect(Value: TSet)
```

Use this method to include all items from the set that also exist in the set passed as the parameter.

TSet.IsEmpty Method

```
function IsEmpty: Boolean
```

Use this method to determine if the set is empty.

TSet.Range Method

```
procedure Range(AStart, AEnd: Integer)
```

Use this method to include a range of items in the set. The AStart parameter indicates the starting item, and the AEnd parameter indicates the ending item.

TSet.Remove Method

```
function Remove(Value: Integer): Boolean
```

Use this method to remove an item from the set.

TSet.Union Method

```
procedure Union(Value: TSet)
```

Use this method to add all items in the set passed as the parameter to the current set.

10.202 TShadow Component

Unit: WebUI

Inherits From TElementAttribute

The TShadow class represents the inset and outset shadows of a UI element or control. Shadows can be a certain color and size, and their placement depends upon whether the shadow is an inset or outset shadow.

Properties	Methods	Events
Blur	SetToDefault	
Color		
HorzOffset		
Spread		
VertOffset		
Visible		

TShadow.Blur Property

```
property Blur: Integer
```

Specifies the amount of blur for the shadow. The amount of blur is equal to the radius, in pixels, of a circular blur transformation that is applied to the solid shadow. This means that the center point of the blur transformation will always be the edge of the solid shadow before the blur is applied.

TShadow.Color Property

```
property Color: TColor
```

Specifies the color of the solid shadow before any blur transformation is applied.

TShadow.HorzOffset Property

```
property HorzOffset: Integer
```

Specifies a horizontal offset, in pixels, to add to the position of the shadow.

TShadow.Spread Property

```
property Spread: Integer
```

Specifies the amount, in pixels, to increase the size of the solid shadow by before any blur transformation is applied.

TShadow.VertOffset Property

```
property VertOffset: Integer
```

Specifies a vertical offset, in pixels, to add to the position of the shadow.

TShadow.Visible Property

```
property Visible: Boolean
```

Specifies whether the shadow is visible.

TShadow.SetToDefault Method

```
procedure SetToDefault
```

Use this method to reset the shadow's properties to their default values.

10.203 TSizeGrip Component

Unit: WebSizer

Inherits From TSizeGripControl

The TSizeGrip component represents a size grip control. A size grip control can be used to allow the user to dynamically size the parent control of the size grip, which is always positioned in the bottom right-hand corner of the parent control.

Properties	Methods	Events
Background		OnClick
InsetShadow		OnDblClick
Opacity		OnHide
		OnMouseDown
		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TSizeGrip.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TSizeGrip.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Specifies the inset shadow for the control.

TSizeGrip.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TSizeGrip.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TSizeGrip.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TSizeGrip.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TSizeGrip.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TSizeGrip.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TSizeGrip.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TSizeGrip.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TSizeGrip.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TSizeGrip.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TSizeGrip.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TSizeGrip.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TSizeGrip.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TSizeGrip.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TSizeGrip.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TSizeGrip.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.204 TSizeGripControl Component

Unit: WebSizer

Inherits From TControl

The TSizeGripControl control is the base class for size grip controls, and contains all of the size grip functionality in the form of public methods and protected properties/events that descendant classes can use to create customized size grip controls.

Properties	Methods	Events

10.205 TSizer Component

Unit: WebSizer

Inherits From TSizerControl

The TSizer component represents a sizer control. A sizer control can be used to allow the user to dynamically size a specific control in a horizontal or vertical orientation.

Properties	Methods	Events
Border		OnAnimationComplete
Control		OnAnimationsComplete
Corners		OnClick
InsetShadow		OnDblClick
Opacity		OnHide
Orientation		OnMouseDown
Padding		OnMouseEnter
		OnMouseLeave
		OnMouseMove
		OnMouseUp
		OnMove
		OnShow
		OnSize
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TSizer.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TSizer.Control Property

```
property Control: TControl
```

Specifies the control to be resized.

TSizer.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TSizer.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Specifies the inset shadow for the control.

TSizer.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TSizer.Orientation Property

```
property Orientation: TSizerOrientation
```

Specifies the orientation of the sizer control, which determines how the sizer control modifies the dimensions of the control specified in the Control property.

TSizer.Padding Property

```
property Padding: TPadding
```

Specifies the padding within the client area of the control.

TSizer.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TSizer.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TSizer.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TSizer.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TSizer.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TSizer.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TSizer.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TSizer.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TSizer.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TSizer.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TSizer.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TSizer.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TSizer.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TSizer.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TSizer.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TSizer.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TSizer.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.206 TSizerControl Component

Unit: WebSizer

Inherits From TControl

The TSizerControl control is the base class for sizer controls, and contains all of the sizer functionality in the form of public methods and protected properties/events that descendant classes can use to create customized sizer controls.

Properties	Methods	Events

10.207 TSlideShow Component

Unit: WebSlide

Inherits From TSlideShowControl

The TSlideShow component represents a control that can display a slideshow of images with configurable transition and animation effects.

Properties	Methods	Events
CacheSize		OnAnimationComplete
Canvas		OnAnimationsComplete
Cursor		OnClick
DisplayTime		OnDblClick
FadeTime		OnHide
Hint		OnLoadSlide
ImageURLs		OnMouseDown
Loop		OnMouseEnter
Opacity		OnMouseLeave
Panning		OnMouseMove
ZoomFactor		OnMouseUp
		OnMove
		OnRenderSlide
		OnShow
		OnSize
		OnStart
		OnStop
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart

TSlideShow.CacheSize Property

```
property CacheSize: Integer
```

Specifies the number of images to cache before the slideshow begins to play. The default value is 2, but you may need to increase this property for low-bandwidth connections in order to avoid issues with playback.

TSlideShow.Canvas Property

```
property Canvas: TCanvasElement
```

This property provdes access to a TCanvasElement instance that can be used to perform drawing operations on the control from within an OnRenderSlide event handler.

TSlideShow.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TSlideShow.DisplayTime Property

```
property DisplayTime: Integer
```

Specifies how long, in milliseconds, each slideshow image should be shown before any transition effects are started for the next image. The default value is 8000.

TSlideShow.FadeTime Property

```
property FadeTime: Integer
```

Specifies how long, in milliseconds, the fade in/fade out effect occurs for each slideshow image transition. The default value is 1000.

TSlideShow.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TSlideShow.ImageURLs Property

```
property ImageURLs: TStrings
```

Specifies the URLs for all images in the slideshow. You must specify at least CacheSize images or an error will occur when trying to play the slideshow using the Start method.

TSlideShow.Loop Property

```
property Loop: Boolean
```

Specifies whether the slideshow should restart with the first image after reaching the last image in the ImageURLs property. The default value is False.

TSlideShow.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TSlideShow.Panning Property

```
property Panning: Boolean
```

Specifies whether the images should be animated so that they pan (move) randomly within the slideshow client area while being displayed. The default value is True.

Note
This property controls one aspect of an effect known as the "Ken Burns Effect", whose name comes from the effects that director Ken Burns made famous with documentaries such as "The Civil War". The ZoomFactor property controls the other aspect, the zooming effect.

TSlideShow.ZoomFactor Property

```
property ZoomFactor: Integer
```

Specifies whether the images should be animated so that they zoom randomly in or out within the slideshow client area while being displayed. The default value is 15, and valid values are 0 (no zooming) to 100 (very fast zooming).

Note

This property controls one aspect of an effect known as the "Ken Burns Effect", whose name comes from the effects that director Ken Burns made famous with documentaries such as "The Civil War". The Panning property controls the other aspect, the panning effect.

TSlideShow.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TSlideShow.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TSlideShow.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TSlideShow.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TSlideShow.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TSlideShow.OnLoadSlide Event

```
property OnLoadSlide: TSlideEvent
```

This event is triggered whenever a new slide image is loaded. Slide images are only loaded once and then cached until the Stop method is called.

TSlideShow.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TSlideShow.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TSlideShow.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TSlideShow.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TSlideShow.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TSlideShow.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TSlideShow.OnRenderSlide Event

```
property OnRenderSlide: TSlideEvent
```

This event is triggered whenever a new slide image is rendered. This event is useful for situations where slide images need to be annotated. Because the TSlideShow control is a direct descendant of the TPaint control, it has a Canvas property that can be used to allow for direct drawing from within any event handlers attached to this event.

Note
This event is triggered FramesPerSecond times per second, so it is very important that any code called from within any event handlers for this event is not time-consuming.

TSlideShow.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TSlideShow.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TSlideShow.OnStart Event

```
property OnStart: TNotifyEvent
```

This event is triggered whenever the Start method is called, and the slide images being rendering.

Note
This event will not be triggered until CacheSize images are loaded and ready to be displayed.

TSlideShow.OnStop Event

```
property OnStop: TNotifyEvent
```

This event is triggered whenever the Stop method is called.

TSlideShow.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TSlideShow.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TSlideShow.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TSlideShow.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

10.208 TSlideShowControl Component

Unit: WebSlide

Inherits From TControl

The TSlideShowControl control is the base class for slideshow controls, and contains all of the slideshow functionality in the form of public methods and protected properties/events that descendant classes can use to create customized slideshow controls.

Properties	Methods	Events
Started	Start	
	Stop	

TSlideShowControl.Started Property

```
property Started: Boolean
```

Indicates whether the slideshow has been started using the Start method.

TSlideShowControl.Start Method

```
procedure Start
```

Use this method to start the slideshow. Once this method is called, the Started property will change to True.

TSlideShowControl.Stop Method

```
procedure Stop
```

Use this method to stop the slideshow. Once this method is called, the Started property will change to False.

10.209 TStateButtonControl Component

Unit: WebBtns

Inherits From TInputControl

The TStateButtonControl control is the base class for button controls that represent selection states, and contains all of the selection state functionality in the form of public methods and protected properties/events that descendant classes can use to create customized selection state button controls.

Properties	Methods	Events

10.210 TStreetViewControlOptions Component

Unit: WebMaps

Inherits From TMapOption

The TStreetViewControlOptions class controls how the street view control is configured in a TMap control. These street view control options correspond to the street view control options available for maps in the Google Maps API.

Properties	Methods	Events
Position		

TStreetViewControlOptions.Position Property

```
property Position: TMapControlPosition
```

Specifies the position of the street view control.

10.211 TStringBuilder Component

Unit: WebCore

Inherits From TObject

The TStringBuilder class is used to manipulate individual characters in a string. Strings in JavaScript (the emitted output from the Elevate Web Builder compiler) are immutable, meaning that you cannot insert, modify, or delete characters in-place within the string. The TStringBuilder class allows for such operations on a string.

Properties	Methods	Events
Chars	Append	
Length	Create	
	GetString	
	Insert	
	Remove	
	ToString	

TStringBuilder.Chars Property

```
property Chars[Index: Integer]: Char
```

Use this property to get access to the individual characters in the string as an array.

TStringBuilder.Length Property

```
property Length: Integer
```

Specifies the length of the string.

TStringBuilder.Append Method

```
procedure Append(const Value: String)

procedure Append(Value: Char)
```

Use this method to append a single character, or another string, to the string.

TStringBuilder.Create Method

```
constructor Create(const Value: String='')
```

Use this method to create a new instance of the TStringBuilder class. The optional Value parameter is used to initialize the string builder instance with a specific string value.

TStringBuilder.GetString Method

```
function GetString(Index: Integer; Count: Integer): String
```

Use this method to retrieve a string containing the specified count of characters from the specified index in the string. If the index plus the count of characters is greater than the total number of characters in the string, then the length of the resultant string will be less than the count that is specified.

TStringBuilder.Insert Method

```
procedure Insert(Position: Integer; const Value: String)

procedure Insert(Position: Integer; Value: Char)
```

Use this method to insert a single character, or another string, into the string at a specific position.

TStringBuilder.Remove Method

```
procedure Remove(Position: Integer; Count: Integer)
```

Use this method to remove one or more characters from the string.

TStringBuilder.ToString Method

```
function ToString: String
```

Use this method to retrieve the actual string value after you are finished manipulating the string.

10.212 TStringList Component

Unit: WebCore

Inherits From TStrings

The TStringList class implements the ancestor TStrings abstract class by providing string storage and sorting/searching functionality.

Properties	Methods	Events
SortCaseInsensitive	Find	
Sorted	Sort	
SortLocaleInsensitive		

TStringList.SortCaseInsensitive Property

```
property SortCaseInsensitive: Boolean
```

Specifies that the list of strings should be sorted in a case-insensitive fashion. The SortCaseInsensitive and SortLocaleInsensitive properties determine how the strings are compared during the sort:

Properties	Function Used

SortCaseInsensitive=False
SortLocaleInsensitive=True
	CompareStr

SortCaseInsensitive=True
SortLocaleInsensitive=True
	CompareText

SortCaseInsensitive=False
SortLocaleInsensitive=False
	LocaleCompareStr

SortCaseInsensitive=True
SortLocaleInsensitive=False
	LocaleCompareText

TStringList.Sorted Property

```
property Sorted: Boolean
```

Specifies that the list of strings is sorted. When the list of strings is sorted, any operations that modify the list automatically trigger a re-sort of the strings in the list.

TStringList.SortLocaleInsensitive Property

```
property SortLocaleInsensitive: Boolean
```

Specifies that the list of strings should be sorted in a locale-insensitive fashion. The SortCaseInsensitive and SortLocaleInsensitive properties determine how the strings are compared during the sort:

Properties	Function Used

SortCaseInsensitive=False
SortLocaleInsensitive=True
	CompareStr

SortCaseInsensitive=True
SortLocaleInsensitive=True
	CompareText

SortCaseInsensitive=False
SortLocaleInsensitive=False
	LocaleCompareStr

SortCaseInsensitive=True
SortLocaleInsensitive=False
	LocaleCompareText

TStringList.Find Method

```
function Find(const Value: String; NearestMatch: Boolean=False):
      Integer
```

Use this method to perform a binary search of the list of strings. The Sorted property must be True or calling this method will result in an exception being raised. The SortCaseInsensitive and SortLocaleInsensitive properties determine how the strings are compared during the search:

Properties	Function Used

SortCaseInsensitive=False
SortLocaleInsensitive=True
	CompareStr

SortCaseInsensitive=True
SortLocaleInsensitive=True
	CompareText

SortCaseInsensitive=False
SortLocaleInsensitive=False
	LocaleCompareStr

SortCaseInsensitive=True
SortLocaleInsensitive=False
	LocaleCompareText

TStringList.Sort Method

```
procedure Sort
```

Use this method to sort the list of strings manually. Normally, the Sorted property would be used to keep the list sorted at all times, but sometimes the developer needs a finer level of control over when the list is sorted. The SortCaseInsensitive and SortLocaleInsensitive properties determine how the strings are compared during the sort:

Properties	Function Used

SortCaseInsensitive=False
SortLocaleInsensitive=True
	CompareStr

SortCaseInsensitive=True
SortLocaleInsensitive=True
	CompareText

SortCaseInsensitive=False
SortLocaleInsensitive=False
	LocaleCompareStr

SortCaseInsensitive=True
SortLocaleInsensitive=False
	LocaleCompareText

10.213 TStrings Component

Unit: WebCore

Inherits From TAbstractList

The TStrings class is an abstract class that is used to manage a list of strings. Because it is an abstract class, the storage of the strings is not implemented in this class, but is rather left to descendant classes to implement. For example, the TStringList descendant class actually contains an internal array that is used to store the list of strings.

Properties	Methods	Events
Count	Add	
LineSeparator	Assign	
Names	Clear	
NameValueSeparator	Delete	
Strings	IndexOf	
Text	IndexOfName	
ValueFromIndex	IndexOfValue	
Values	Insert	
	Remove	

TStrings.Count Property

```
property Count: Integer
```

Indicates the total number of strings in the list.

TStrings.LineSeparator Property

```
property LineSeparator: String
```

Specifies the character, or characters, to use to separate multiple strings into the list of strings when assigning a string value to the Text property. These characters are also used to format a string when reading the Text property. The default value for this property is the carriage return (0x0D) character and linefeed (0x0A) character.

Note
In the JavaScript runtime environment, the linefeed character is sufficient to express a line break, so controls like the TMultiLineEdit component use a LineSeparator of just a linefeed character for its Lines property.

TStrings.Names Property

```
property Names[Index: Integer]: String
```

Allows access to the name portion of name/value pairs in the list of strings. The index specifies the position of the name in the list that you wish to access or assign.

TStrings.NameValueSeparator Property

```
property NameValueSeparator: Char
```

Specifies the character, or characters, to use to separate name/value pairs in the list of strings. The Names, ValueFromIndex, and Values properties can be used to work wtih the name/value pairs. The default value for this property is the equals (=) character.

TStrings.Strings Property

```
property Strings[Index: Integer]: String
```

Allows indexed access to all strings in the list.

TStrings.Text Property

```
property Text: String
```

Specifies the list of strings as one formatted string. The LineSeparator property is used to format the strings in the list into one string.

TStrings.ValueFromIndex Property

```
property ValueFromIndex[Index: Integer]: String
```

Allows access to the value portion of name/value pairs in the list of strings. The index specifies the position of the value in the list that you wish to access or assign.

TStrings.Values Property

```
property Values[const Name: String]: String
```

Allows access to the value portion of name/value pairs in the list of strings. The name specifies the name portion of the name/value pair in the list that you wish to access or assign a value for.

TStrings.Add Method

```
function Add(const Value: String): Integer
```

Use this method to add a string to the end of the list of strings. The string will be added to the end of the list, and the index of the string in the list will be returned.

TStrings.Assign Method

```
procedure Assign(Value: TStrings)
```

Use this method to clear the list of strings and replace them with a copy of the list of strings passed as the parameter.

TStrings.Clear Method

```
procedure Clear
```

Use this method to remove all strings from the list.

TStrings.Delete Method

```
procedure Delete(Index: Integer)
```

Use this method to remove a string from the list of strings using its index.

TStrings.IndexOf Method

```
function IndexOf(const Value: String; StartIndex: Integer=0;
      PartialMatch: Boolean=False): Integer
```

Use this method to return the index of a particular string in the list of strings. The StartIndex parameter indicates the index into the list of strings at which to start the search, and the PartialMatch parameter indicates whether only the length of the search string should be used when performing the comparisons. The SameText function is used by this method to compare the strings.

TStrings.IndexOfName Method

```
function IndexOfName(const Name: String; StartIndex: Integer=0;
      PartialMatch: Boolean=False): Integer
```

Use this method to return the index of a particular name portion of the name/value pairs in the list of strings. The SameText function is used by this method to compare the names.

TStrings.IndexOfValue Method

```
function IndexOfValue(const Value: String): Integer
```

Use this method to return the index of a particular value portion of the name/value pairs in the list of strings. The SameText function is used by this method to compare the values.

TStrings.Insert Method

```
procedure Insert(Index: Integer; const Value: String)
```

Use this method to insert a string at a specific index in the list of strings.

TStrings.Remove Method

```
function Remove(const Value: String): Integer
```

Use this method to remove a string from the list of strings by its value. The IndexOf method is used by this method to find the string.

10.214 TStringValue Component

Unit: WebCore

Inherits From TDataValue

This class represents the value for a String column in a row in a TDataSet component.

Properties	Methods	Events

10.215 TSurface Component

Unit: WebForms

Inherits From TScrollableControl

The TSurface component represents the application surface that is included with every visual application and provides properties and methods for iterating through all forms in the application, specifying the layout of the application surface, and showing message and progress dialogs.

Properties	Methods	Events
ActiveForm	HideProgress	OnSize
Background	MessageDlg	
Cursor	ShowMessage	
MaxDialogWidth	ShowProgress	
ModalOverlay		
ScrollBars		
ScrollSupport		

TSurface.ActiveForm Property

```
property ActiveForm: TFormControl
```

Indicates the active TFormControl instance in the application.

TSurface.Background Property

```
property Background: TBackground
```

Specifies the background for the control.

TSurface.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TSurface.MaxDialogWidth Property

```
property MaxDialogWidth: Integer
```

Specifies the maximum dialog width to use for system-generated message dialogs. The default value is 50% of the available width of the application viewport.

With mobile applications, it may be desirable to adjust this property to allow for full-screen message dialogs that work better with narrow portrait orientations.

TSurface.ModalOverlay Property

```
property ModalOverlay: TModalOverlay
```

Provides access to the modal overlay instance for the application's surface.

TSurface.ScrollBars Property

```
property ScrollBars: TScrollBars
```

Specifies which scrollbars to show, if any.

Note
Even if this property is set to sbHorizontal, sbVertical, or sbBoth, a scrollbar will only be shown if the size of the contents and/or the child controls of the control exceed the client rectangle for the control.

TSurface.ScrollSupport Property

```
property ScrollSupport: TScrollSupport
```

Specifies the directions in which the control can be scrolled, if any.

> **Note**
> This property only applies to scrolling via touch and mouse wheel movements, and does not apply to programmatic scrolling or scrolling via the scroll bars themselves.

TSurface.HideProgress Method

```
procedure HideProgress
```

Use this method to decrement the global progress reference count, and if the reference count is 0, hide the active progress dialog. The ShowProgress method shows a progress dialog and increments the progress reference count.

TSurface.MessageDlg Method

```
procedure MessageDlg(const Msg: String; const DlgCaption: String;
       DlgType: TMsgDlgType; const Buttons: TMsgDlgBtns;
      DefaultButton: TMsgDlgBtn; MsgDlgResult: TMsgDlgResultEvent=nil;
      DlgClose: Boolean=False; AnimationStyle: TAnimationStyle=asNone;
      AnimationDuration: Integer=0)
```

Use this method to display a modal message dialog. Please see the MessageDlg procedure for more information on the parameters to this method.

TSurface.ShowMessage Method

```
procedure ShowMessage(const Msg: String; const DlgCaption:
      String=''; AnimationStyle: TAnimationStyle=asNone;
      AnimationDuration: Integer=0)
```

Use this method to show a simple modal message dialog. The Msg parameter indicates the message to show.

TSurface.ShowProgress Method

```
procedure ShowProgress(const Msg: String; AnimationStyle:
      TAnimationStyle=asNone; AnimationDuration: Integer=0)
```

Use this method to show a modal progress dialog and increment the global progress reference count. The HideProgress method decrements the reference count and hides any active progress dialog.

TSurface.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

10.216 TTab Component

Unit: WebPages

Inherits From TControl

The TTab component represents the tab for a TPage instance within a TPagePanel control. Each page instance contains a reference to a tab control via its Tab property. The properties in the tab can be modified to affect the tab caption and how the tab is sized and formatted.

Properties	Methods	Events
AllowClose		
AutoWidth		
Caption		
Font		
Hint		

TTab.AllowClose Property

```
property AllowClose: Boolean
```

Specifies whether the close button should be shown in the tab.

TTab.AutoWidth Property

```
property AutoWidth: Boolean
```

Specifies whether the width of the tab should be automatically set based upon the Caption and Font properties.

TTab.Caption Property

```
property Caption: String
```

Specifies the caption to display on the tab.

TTab.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TTab.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

10.217 TTextAreaElement Component

Unit: WebUI

Inherits From TInputElement

The TTextAreaElement class is the base element class for text area elements, and contains all of the text area functionality in the form of public methods and properties/events that control classes can use to create text area controls.

> **Note**
> This element does not provide support for text area elements at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events

10.218 TTextInputElement Component

Unit: WebUI

Inherits From TInputElement

The TTextInputElement class is the base element class for text input elements, and contains all of the text input functionality in the form of public methods and properties/events that control classes can use to create text input controls.

> **Note**
> This element does not provide support for text input elements at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events
InputType		

TTextInputElement.InputType Property

```
property InputType: TTextInputType
```

Specifies the type of text being input into the element by the user. This information is used by the browser to determine how to display and edit the text. For example, in touch environments, this property is used to determine which soft keyboard to display to the user.

10.219 TTimer Component

Unit: WebComps

Inherits From TComponent

The TTimer component represents a timer that triggers an OnTimer event whenever the interval, specified in milliseconds, elapses. Timers are asynchronous, meaning that they can trigger the OnTimer event even while the user is performing other tasks in the web browser.

Properties	Methods	Events
Enabled		OnTimer
Interval		

TTimer.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the timer is enabled. Whenever the timer is enabled, the Interval for the timer starts from zero. The default value is True.

TTimer.Interval Property

```
property Interval: Integer
```

Specifies the interval for the timer in milliseconds. Whenever the interval elapses, the OnTimer event is triggered. The default value is 1000 milliseconds.

TTimer.OnTimer Event

```
property OnTimer: TNotifyEvent
```

This event is triggered whenever the Interval specified for the timer elapses.

10.220 TTimeValue Component

Unit: WebCore

Inherits From TDateTimeValue

This class represents the value for a Time column in a row in a TDataSet component.

Properties	Methods	Events

10.221 TToolBar Component

Unit: WebTlbrs

Inherits From TToolBarControl

The TToolBar class represents a toolbar control. A toolbar control contains 0 or more TToolBarButton instances that serve as non-focusable buttons, and is ideal for menus.

Properties	Methods	Events
Background		OnAnimationComplete
Border		OnAnimationsComplete
Corners		OnButtonClick
Cursor		OnHide
Hint		OnMove
InsetShadow		OnShow
MultiSelect		OnSize
Opacity		

TToolBar.Background Property

```
property Background: TBackground
```

Specifies the background of the control.

TToolBar.Border Property

```
property Border: TBorder
```

Specifies the border for the control.

TToolBar.Corners Property

```
property Corners: TCorners
```

Specifies the horizontal and vertical radii for the corners of the control.

TToolBar.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TToolBar.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TToolBar.InsetShadow Property

```
property InsetShadow: TInsetShadow
```

Specifies the inset shadow for the control.

TToolBar.MultiSelect Property

```
property MultiSelect: Boolean
```

Specifies whether multiple toolbar buttons can have their Selected property set to True at the same time. The default value is True.

Note
Toolbar buttons can only be selected if their AllowSelection property is set to True.

TToolBar.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TToolBar.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TToolBar.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TToolBar.OnButtonClick Event

```
property OnButtonClick: TClickEvent
```

This event is triggered whenever one of the toolbar buttons is clicked.

TToolBar.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TToolBar.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TToolBar.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TToolBar.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

10.222 TToolBarButton Component

Unit: WebTlbrs

Inherits From TRepeatControl

The TToolBarButton class represents a button on a TToolBar control. Toolbar buttons consist of an icon and an optional caption, and cannot obtain focus.

Properties	Methods	Events
AllowSelection		OnClick
AutoWidth		OnHide
Caption		OnShow
Cursor		
Enabled		
Font		
Hint		
Icon		
Index		
ParentToolBar		
RepeatClick		
RepeatClickInterval		
Selected		

TToolBarButton.AllowSelection Property

```
property AllowSelection: Boolean
```

Specifies whether the toolbar button is selectable. If a toolbar button is selectable, then its Selected property can be modified to toggle the button to and from a "pushed" state.

Note

By default, multiple toolbar buttons within the same toolbar can have their Selected property set to True at the same time. This behavior is controlled by the TToolBar MultiSelect property. If the MultiSelect property is set to True (the default), then the toolbar buttons will behave like check boxes. If the MultiSelect property is set to False, then the toolbar buttons will behave like radio buttons.

TToolBarButton.AutoWidth Property

```
property AutoWidth: Boolean
```

Specifies whether the width of the button should be automatically set based upon the Caption, Icon, and Font properties.

TToolBarButton.Caption Property

```
property Caption: String
```

Specifies the caption to display on the button.

TToolBarButton.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TToolBarButton.Enabled Property

```
property Enabled: Boolean
```

Specifies whether the button is enabled or disabled. When a button is disabled, it cannot be clicked and is displayed in a disabled state. The default value is True.

TToolBarButton.Font Property

```
property Font: TFont
```

Specifies the properties of the font used to display the content of the control.

TToolBarButton.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TToolBarButton.Icon Property

```
property Icon: TIconProperties
```

Specifies the properties of the icon used with the control.

TToolBarButton.Index Property

```
property Index: Integer
```

Specifies the index of the button in its parent toolbar's buttons.

TToolBarButton.ParentToolBar Property

```
property ParentToolBar: TToolBarControl
```

Indicates the parent toolbar that contains the button.

TToolBarButton.RepeatClick Property

```
property RepeatClick: Boolean
```

Specifies whether the OnClick event handler should be triggered every RepeatClickInterval milliseconds while the button is pressed.

TToolBarButton.RepeatClickInterval Property

```
property RepeatClickInterval: Integer
```

Specifies the interval, in milliseconds, to trigger the OnClick event handler when the RepeatClick is True and the button is pressed.

TToolBarButton.Selected Property

```
property Selected: Boolean
```

Specifies whether the toolbar button is selected. A toolbar button can only be selected if the AllowSelection property is True. If a toolbar button is selectable, then the Selected property can be modified to toggle the button to and from a "pushed" state.

Note
By default, multiple toolbar buttons within the same toolbar can have their Selected property set to True at the same time. This behavior is controlled by the TToolBar MultiSelect property. If the MultiSelect property is set to True (the default), then the toolbar buttons will behave like check boxes. If the MultiSelect property is set to False, then the toolbar buttons will behave like radio buttons.

TToolBarButton.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TToolBarButton.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TToolBarButton.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

10.223 TToolBarControl Component

Unit: WebTlbrs

Inherits From TControl

The TToolBarControl control is the base class for toolbar controls, and contains all of the toolbar functionality in the form of public methods and protected properties/events that descendant classes can use to create customized toolbar controls.

Properties	Methods	Events
ButtonCount	MakeButtonVisible	
Buttons	NewButton	
VisibleButtonCount	ScrollNext	
VisibleButtons	ScrollPrior	

TToolBarControl.ButtonCount Property

```
property ButtonCount: Integer
```

Indicates the number of buttons in the toolbar control.

TToolBarControl.Buttons Property

```
property Buttons[AIndex: Integer]: TToolBarButton
```

Accesses the buttons in the toolbar control by index.

TToolBarControl.VisibleButtonCount Property

```
property VisibleButtonCount: Integer
```

Indicates the number of visible buttons in the toolbar control.

TToolBarControl.VisibleButtons Property

```
property VisibleButtons[AIndex: Integer]: TToolBarButton
```

Accesses the visible buttons in the toolbar control by index.

TToolBarControl.MakeButtonVisible Method

```
procedure MakeButtonVisible(AButton: TToolBarButton)
```

Use this method to ensure that the specified button is visible.

TToolBarControl.NewButton Method

```
function NewButton: TToolBarButton
```

Use this method to create a new button. The new button will be positioned after all other existing buttons.

TToolBarControl.ScrollNext Method

```
procedure ScrollNext
```

Use this method to scroll the buttons to the left so that the left-most button in the control is no longer visible.

TToolBarControl.ScrollPrior Method

```
procedure ScrollPrior
```

Use this method to scroll the buttons to the right so that the button to the left of the left-most button in the control is visible.

10.224 TVideo Component

Unit: WebMedia

Inherits From TMediaControl

The TVideo control encapsulates the HTML5 video support available in web browsers. With the TVideo control, you can handle most aspects of video loading and playback.

Properties	Methods	Events
AutoPlay		OnAbort
CurrentTime		OnAnimationComplete
Cursor		OnAnimationsComplete
DataColumn		OnCanPlay
DataSet		OnCanPlayThrough
DefaultPlaybackRate		OnClick
Duration		OnDblClick
Ended		OnDurationChange
Hint		OnEmptied
Loop		OnEnded
Muted		OnError
NetworkState		OnHide
Opacity		OnLoadedData
Paused		OnLoadedMetadata
PlaybackRate		OnLoadStart
PosterImageURL		OnMouseDown
Preload		OnMouseEnter
ReadyState		OnMouseLeave
Seeking		OnMouseMove
ShowControls		OnMouseUp
SourceURL		OnMove
VideoHeight		OnPause
VideoWidth		OnPlay
Volume		OnPlaying
		OnProgress
		OnRateChange

		OnSeeked
		OnSeeking
		OnShow
		OnSize
		OnStalled
		OnSuspend
		OnTimeUpdate
		OnTouchCancel
		OnTouchEnd
		OnTouchMove
		OnTouchStart
		OnVolumeChange
		OnWaiting

TVideo.AutoPlay Property

```
property AutoPlay: Boolean
```

Specifies that the video should begin playing as soon as enough data has been loaded to allow playback. The default value is False.

TVideo.CurrentTime Property

```
property CurrentTime: Double
```

Indicates the current playback time, in seconds. Setting this property to a new value will cause the video to skip to the specified time.

TVideo.Cursor Property

```
property Cursor: TCursor
```

Specifies the cursor to use when the mouse hovers over the control. The default value is crAuto.

TVideo.DataColumn Property

```
property DataColumn: String
```

Specifies the data column name to bind to in the dataset specified by the DataSet property. The default value is ''.

TVideo.DataSet Property

```
property DataSet: TDataSet
```

Specifies the dataset to bind the control to. The default value is nil.

TVideo.DefaultPlaybackRate Property

```
property DefaultPlaybackRate: Double
```

Specifies the default playback rate, with 1 being normal playback, less than 1 being slower playback, and greater than 1 being faster playback. The default value is 1.

Note
The volume will normally be automatically muted when playing video faster or slower than the normal playback rate.

TVideo.Duration Property

```
property Duration: Double
```

Indicates the length of the video in seconds. Add an event handler for the OnDurationChange event to detect when the duration has been determined for the current video being loaded/played. If the duration has not been determined, this property will return 0.

TVideo.Ended Property

```
property Ended: Boolean
```

Indicates that the end of the video has been reached.

TVideo.Hint Property

```
property Hint: String
```

Specifies the hint to display in the web browser when the mouse hovers over the control for a browser-specific amount of time. The default value is ''.

TVideo.Loop Property

```
property Loop: Boolean
```

Specifies that the video playback should automatically restart at the beginning once the end has been reached. The default value is False.

TVideo.Muted Property

```
property Muted: Boolean
```

Specifies that the playback volume should be muted. The default value is False.

TVideo.NetworkState Property

```
property NetworkState: TMediaNetworkState
```

Indicates the network state of the video loading/playback.

TVideo.Opacity Property

```
property Opacity: Integer
```

Specifies the opacity of the control, with the valid values being 0 (transparent) to 100 (completely opaque). The default value is 100.

TVideo.Paused Property

```
property Paused: Boolean
```

Indicates that video playback is paused, either by the user pausing the video via the user interface when the ShowControls property is True, or by the application calling the Pause method. The default value is False.

TVideo.PlaybackRate Property

```
property PlaybackRate: Double
```

Specifies the playback rate, with 1 being normal playback, less than 1 being slower playback, and greater than 1 being faster playback. The default value is 1.

Note
 The volume will normally be automatically muted when playing video faster or slower than the normal playback rate.

TVideo.PosterImageURL Property

```
property PosterImageURL: String
```

Specifies the URL of an image to use as a background before video playback is started.

TVideo.Preload Property

```
property Preload: TMediaPreload
```

Specifies how much of the current video data should be loaded before playback begins.

TVideo.ReadyState Property

```
property ReadyState: TMediaReadyState
```

Indicates whether the video is ready for playback, and if so, a general description of what video data has been loaded.

TVideo.Seeking Property

```
property Seeking: Boolean
```

Indicates that video is switching to a new playback location, either by the user changing the playback location in the video via the user interface when the ShowControls property is True, or by the application setting the CurrentTime property.

TVideo.ShowControls Property

```
property ShowControls: Boolean
```

Specifies whether the control should show the native user interface for the video being played.

TVideo.SourceURL Property

```
property SourceURL: String
```

Specifies the URL of the video to be loaded into the control. Whenever this property is changed, the existing video is cleared and the new video will start downloading from the web server. Please review the events available for this control in order to get more information on detecting and handling the loading/playback of the video.

TVideo.VideoHeight Property

```
property VideoHeight: Integer
```

Indicates the actual height of the video being played.

Note
This property will be zero until the metadata for the video has been loaded.

TVideo.VideoWidth Property

```
property VideoWidth: Integer
```

Indicates the actual width of the video being played.

Note
This property will be zero until the metadata for the video has been loaded.

TVideo.Volume Property

```
property Volume: Integer
```

Specifies the playback volume of the audio for the video. The volume can be set between 0 and 100.

TVideo.OnAbort Event

```
property OnAbort: TNotifyEvent
```

This event is triggered whenever the media control has stopped loading data for the current media. This is normally caused by the user requesting such an action via the user interface when the ShowControls property is True.

TVideo.OnAnimationComplete Event

```
property OnAnimationComplete: TAnimationCompleteEvent
```

This event is triggered when an animation completes for the control.

TVideo.OnAnimationsComplete Event

```
property OnAnimationsComplete: TNotifyEvent
```

This event is triggered when all active animations complete for the control.

TVideo.OnCanPlay Event

```
property OnCanPlay: TNotifyEvent
```

This event is triggered whenever the media control has loaded enough data to begin playback. However, additional data loading may be required as playback continues.

TVideo.OnCanPlayThrough Event

```
property OnCanPlayThrough: TNotifyEvent
```

This event is triggered whenever the media control has loaded enough data to begin playback and (probably) play the media until the end without needing to load any additional data.

TVideo.OnClick Event

```
property OnClick: TNotifyEvent
```

This event is triggered when the control is clicked with the mouse pointer or is tapped using a touch interface.

TVideo.OnDblClick Event

```
property OnDblClick: TNotifyEvent
```

This event is triggered when the control is double-clicked with the mouse pointer or is double-tapped using a touch interface.

TVideo.OnDurationChange Event

```
property OnDurationChange: TNotifyEvent
```

This event is triggered whenever the duration of the media changes, which normally occurs when loading new media into the control by modifying the SourceURL property.

TVideo.OnEmptied Event

```
property OnEmptied: TNotifyEvent
```

This event is triggered whenever an error or abort has caused the NetworkState property to revert to the mnsEmpty state.

TVideo.OnEnded Event

```
property OnEnded: TNotifyEvent
```

This event is triggered whenever playback has stopped because the end of the media has been reached.

TVideo.OnError Event

```
property OnError: TNotifyEvent
```

This event is triggered whenever an error has prevented the media from being loaded properly.

TVideo.OnHide Event

```
property OnHide: TNotifyEvent
```

This event is triggered when the control is hidden using the Hide method.

TVideo.OnLoadedData Event

```
property OnLoadedData: TNotifyEvent
```

This event is triggered whenever the media control has loaded enough data for the current playback location.

TVideo.OnLoadedMetadata Event

```
property OnLoadedMetadata: TNotifyEvent
```

This event is triggered whenever the media control has loaded the metadata, including the duration and dimensions, for the current media.

TVideo.OnLoadStart Event

```
property OnLoadStart: TNotifyEvent
```

This event is triggered whenever the media control starts loading data for the current media.

TVideo.OnMouseDown Event

```
property OnMouseDown: TMouseDownEvent
```

This event is triggered when a mouse button is pressed while the mouse pointer hovers over the control.

TVideo.OnMouseEnter Event

```
property OnMouseEnter: TNotifyEvent
```

This event is triggered when the mouse pointer enters the bounds of the control.

TVideo.OnMouseLeave Event

```
property OnMouseLeave: TNotifyEvent
```

This event is triggered when the mouse pointer leaves the bounds of the control.

TVideo.OnMouseMove Event

```
property OnMouseMove: TMouseMoveEvent
```

This event is triggered as the mouse pointer is moved over the control.

TVideo.OnMouseUp Event

```
property OnMouseUp: TMouseUpEvent
```

This event is triggered when a mouse button is released while the mouse pointer hovers over the control.

TVideo.OnMove Event

```
property OnMove: TNotifyEvent
```

This event is triggered whenever the control's position is changed.

TVideo.OnPause Event

```
property OnPause: TNotifyEvent
```

This event is triggered whenever the media playback is paused.

TVideo.OnPlay Event

```
property OnPlay: TNotifyEvent
```

This event is triggered whenever the media playback is started/resumed.

TVideo.OnPlaying Event

```
property OnPlaying: TNotifyEvent
```

This event is triggered whenever media playback has actually started.

Note

This event is slightly different from the OnPlay event, which only indicates that the user or application **requested** playback to start/resume. This event may be triggered multiple times during playback, especially if playback needs to stop in order to allow more media data to be loaded, which can be the case with slower network connections.

TVideo.OnProgress Event

```
property OnProgress: TNotifyEvent
```

This event is triggered whenever the current media is being loaded.

Note
This event is typically fired several times per second in most web browsers, so be very careful about how time-consuming any event handlers are for this event.

TVideo.OnRateChange Event

```
property OnRateChange: TNotifyEvent
```

This event is triggered whenever the playback rate of the media control has changed for the current media. This is caused by the user requesting such an action via the user interface when the ShowControls property is True, or when the application modifies the PlaybackRate property.

TVideo.OnSeeked Event

```
property OnSeeked: TNotifyEvent
```

This event is triggered whenever the Seeking property reverts to False.

TVideo.OnSeeking Event

```
property OnSeeking: TNotifyEvent
```

This event is triggered whenever the playback location of the media control has changed for the current media. This is caused by the user requesting such an action via the user interface when the ShowControls property is True, or when the application modifies the CurrentTime property.

TVideo.OnShow Event

```
property OnShow: TNotifyEvent
```

This event is triggered when the control is shown using the Show method.

TVideo.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the control's width and/or height are changed.

TVideo.OnStalled Event

```
property OnStalled: TNotifyEvent
```

This event is triggered whenever the media control is trying to load data for the current media, but no data is arriving over the network.

TVideo.OnSuspend Event

```
property OnSuspend: TNotifyEvent
```

This event is triggered whenever the media control has loaded enough data to enable playback, and has stopped loading more data.

TVideo.OnTimeUpdate Event

```
property OnTimeUpdate: TNotifyEvent
```

This event is triggered whenever the CurrentTime property changes.

Note
This event can be fired as many as 60 times per second in some web browsers, so be very careful about how time-consuming any event handlers are for this event.

TVideo.OnTouchCancel Event

```
property OnTouchCancel: TTouchEvent
```

This event is triggered when the browser cancels touching via a touch interface. This can happen, for example, when the touch has exceeded the browser viewport.

TVideo.OnTouchEnd Event

```
property OnTouchEnd: TTouchEvent
```

This event is triggered when the control stops being touched via a touch interface.

TVideo.OnTouchMove Event

```
property OnTouchMove: TTouchEvent
```

This event is triggered as a touch is moved over the control.

TVideo.OnTouchStart Event

```
property OnTouchStart: TTouchEvent
```

This event is triggered when the control is touched via a touch interface.

TVideo.OnVolumeChange Event

```
property OnVolumeChange: TNotifyEvent
```

This event is triggered whenever the audio volume of the media control has changed for the current media. This is caused by the user requesting such an action via the user interface when the ShowControls property is True, or when the application modifies the Volume property.

TVideo.OnWaiting Event

```
property OnWaiting: TNotifyEvent
```

This event is triggered whenever the media control cannot start/resume playback because more data needs to be loaded for the current media.

10.225 TVideoElement Component

Unit: WebUI

Inherits From TMediaElement

The TVideoElement class is the element class for video UI elements, and contains all of the video playback functionality in the form of public methods and properties/events that control classes can use to create video controls.

Note
This element does not provide support for video playback at design-time, and the applicable playback methods and properties are all stubs.

Properties	Methods	Events
PosterURL		
VideoHeight		
VideoWidth		

TVideoElement.PosterURL Property

```
property PosterURL: String
```

Specifies the URL of an image to use as a background before video playback is started.

TVideoElement.VideoHeight Property

```
property VideoHeight: Integer
```

Indicates the actual height of the video being played.

Note
This property will be zero until the metadata for the video has been loaded.

TVideoElement.VideoWidth Property

```
property VideoWidth: Integer
```

Indicates the actual width of the video being played.

Note
This property will be zero until the metadata for the video has been loaded.

10.226 TViewport Component

Unit: WebForms

Inherits From TComponent

The TViewport component represents the browser viewport at run-time for a visual application and provides properties for retrieving the browser viewport dimensions, as well as specifying whether browser scrollbars should be displayed when the dimensions of the application surface overflows the viewport's client area.

Properties	Methods	Events
Height	ScrollBy	OnScroll
OverflowX		OnSize
OverflowY		
ResizeDelay		
ScrollLeft		
ScrollTop		
Width		

TViewport.Height Property

```
property Height: Integer
```

Indicates the height of the browser viewport.

TViewport.OverflowX Property

```
property OverflowX: TOverflowType
```

Specifies whether or not to show a native horizontal browser scrollbar for the application if the width of its surface exceeds the width of the client rectangle for the element.

TViewport.OverflowY Property

```
property OverflowY: TOverflowType
```

Specifies whether or not to show a native vertical browser scrollbar for the application if the height of its surface exceeds the height of the client rectangle for the element.

TViewport.ResizeDelay Property

```
property ResizeDelay: Integer
```

Specifies how long, in milliseconds, the application will wait after a browser viewport resize before updating the application's user interface.

TViewport.ScrollLeft Property

```
property ScrollLeft: Integer
```

If an application's Surface width and/or height is greater than the application viewport's Width and Height properties, then this property indicates the amount, in pixels, that the browser viewport has been scrolled to the right.

Specify a new value to manually scroll the browser viewport to the left or right. A value of 0 means that the viewport is scrolled all the way to its left-most position.

TViewport.ScrollTop Property

```
property ScrollTop: Integer
```

If an application's Surface width and/or height is greater than the application viewport's Width and Height properties, then this property indicates the amount, in pixels, that the browser viewport has been scrolled towards the bottom.

Specify a new value to manually scroll the browser viewport towards the top or bottom. A value of 0 means that the viewport is scrolled all the way to its top-most position.

TViewport.Width Property

```
property Width: Integer
```

Indicates the width of the browser viewport.

TViewport.ScrollBy Method

```
procedure ScrollBy(X,Y: Integer)
```

If an application's Surface width and/or height is greater than the application viewport's Width and Height properties, then you can use this method to scroll the viewport of the application horizontally, vertically, or both. The X and Y values represent the number of pixels to scroll the viewport by, and may be negative values for scrolling backward.

TViewport.OnScroll Event

```
property OnScroll: TNotifyEvent
```

This event is triggered whenever the browser viewport is scrolled horizontally or vertically.

TViewport.OnSize Event

```
property OnSize: TNotifyEvent
```

This event is triggered whenever the browser viewport's width and/or height are changed.

10.227 TWebControl Component

Unit: WebBrwsr

Inherits From TBindableColumnControl

The TWebControl control is the base class for web browser controls that dynamically load resources, and contains all of the web browser control functionality in the form of public methods and protected properties/events that descendant classes can use to create customized web browser controls.

Properties	Methods	Events
	Refresh	

TWebControl.Refresh Method

```
procedure Refresh
```

Use this method to refresh a resource without having to modify the URL. This is useful when the resource changes on the server, but the URL does not.

10.228 TWebElement Component

Unit: WebUI

Inherits From TElement

The TWebElement class is the base element class for browser objects that can be dynamically-loaded as a resource, and contains all of the browser object functionality in the form of public methods and properties/events that control classes can use to create browser object controls.

Note
This element does not provide support for browser objects at design-time, and the applicable methods and properties are all stubs.

Properties	Methods	Events
Loaded	Refresh	OnError
URL		OnLoad
		OnUnload

TWebElement.Loaded Property

```
property Loaded: Boolean
```

Indicates whether the object resource specified by the URL property has been loaded.

An event handler can be attached to the OnLoad event to execute code when the object is loaded.

TWebElement.URL Property

```
property URL: String
```

Specifies the URL for the object resource. Whenever the URL property changes, the OnUnload event is triggered immediately. The OnLoad event is triggered once the object resource has been loaded.

TWebElement.Refresh Method

```
procedure Refresh
```

Use this method to refresh a resource without having to modify the URL property. This is useful when the resource changes on the server, but the URL does not.

TWebElement.OnError Event

```
property OnError: TWebElementEvent
```

This event is triggered an error occurs while loading the resource specified by the URL property.

TWebElement.OnLoad Event

```
property OnLoad: TWebElementEvent
```

This event is triggered when the resource specified by the URL property has been completely loaded.

TWebElement.OnUnload Event

```
property OnUnload: TWebElementEvent
```

This event is triggered when the currently-loaded resource specified by the URL property has been unloaded.

10.229 TWriter Component

Unit: WebCore

Inherits From TObject

The TWriter class is a class used by TPersistent-descendant classes to save their published properties to JSON strings. It can be used as a general-purpose JSON writer in your applications.

When a TWriter instance is created, the constructor allows you to specify:

- The date-time format to use when writing date-time properties.
- The number of spaces to use per indentation level, if the output is not compressed.
- Whether to include line feeds in the JSON output, if the output is not compressed.
- Whether to compress all whitespace in the JSON output. Compressing the whitespace removes any unnecessary whitespace in order to keep the size of the JSON output to a minimum.

Properties	Methods	Events
Output	BeginArray	
	BeginNewLine	
	BeginObject	
	BooleanProperty	
	BooleanValue	
	CancelNewLine	
	Create	
	DateTimeProperty	
	DateTimeValue	
	DecIndent	
	EndArray	
	EndObject	
	FloatProperty	
	FloatValue	
	IncIndent	
	Initialize	
	IntegerProperty	
	IntegerValue	
	Literal	
	NewLine	
	NullProperty	
	NullValue	
	ObjectProperty	
	PropertyName	
	Separator	
	StringProperty	
	StringValue	
	Whitespace	

TWriter.Output Property

```
property Output: String
```

Indicates the current JSON output.

TWriter.BeginArray Method

```
procedure BeginArray(HasElements: Boolean)
```

Use this method to begin writing a new array.

TWriter.BeginNewLine Method

```
procedure BeginNewLine
```

Use this method to set a flag requesting that the next property should be written on a new line.

Note
New lines are not used if the JSON output is being compressed.

TWriter.BeginObject Method

```
procedure BeginObject
```

Use this method to begin writing a new object.

TWriter.BooleanProperty Method

```
procedure BooleanProperty(const Name: String; Value: Boolean)
```

Use this method to write a boolean property.

TWriter.BooleanValue Method

```
procedure BooleanValue(Value: Boolean)
```

Use this method to write a boolean value.

TWriter.CancelNewLine Method

```
procedure CancelNewLine
```

Use this method to cancel a new line started via the NewLine method. This is useful if a new line is started for a property, but the property cannot be written for some reason.

TWriter.Create Method

```
constructor Create(ADateTimeFormat: TDateTimeFormat=dtfRaw;
      AIndentSpaces: Integer=3; AIncludeLineFeeds: Boolean=True;
      ACompressWhitespace: Boolean=False)
```

Use this method to create a new instance of the TWriter class. The optional ADateTimeFormat parameter indicates whether date and time values should be output as an ISO 8601 date and time string value, or as a raw Unix date and time integer value (the number of milliseconds since midnight, January 1, 1970). The optional AIndentSpaces parameter indicates how many spaces to use for indentation in the output, the optional AIncludeLineFeeds parameter indicates whether to include line feeds (CRLF) in the output, and the optional ACompressWhitespace parameter indicates whether any whitespace should be compressed (removed) from the output.

TWriter.DateTimeProperty Method

```
procedure DateTimeProperty(const Name: String; Value: DateTime)
```

Use this method to write a date-time property. How a date-time property is written is controlled by the first TDateTimeFormat parameter in the TWriter class constructor.

TWriter.DateTimeValue Method

```
procedure DateTimeValue(Value: DateTime)
```

Use this method to write a date-time value. How a date-time value is written is controlled by the first TDateTimeFormat parameter in the TWriter class constructor.

TWriter.DecIndent Method

```
procedure DecIndent
```

Decrement the indentation level for the output.

Note
Indentation levels are not used if the JSON output is being compressed.

TWriter.EndArray Method

```
procedure EndArray(HasElements: Boolean)
```

Use this method to end writing an array.

TWriter.EndObject Method

```
procedure EndObject
```

Use this method to end writing an object.

TWriter.FloatProperty Method

```
procedure FloatProperty(const Name: String; Value: Double)
```

Use this method to write a float property.

TWriter.FloatValue Method

```
procedure FloatValue(Value: Double)
```

Use this method to write a float value.

TWriter.IncIndent Method

```
procedure IncIndent
```

Increment the indentation level for the JSON output.

Note
Indentation levels are not used if the JSON output is being compressed.

TWriter.Initialize Method

```
procedure Initialize
```

Use this method to initialize the writer so that the Output property is blank.

TWriter.IntegerProperty Method

```
procedure IntegerProperty(const Name: String; Value: Integer)
```

Use this method to write an integer property.

TWriter.IntegerValue Method

```
procedure IntegerValue(Value: Integer)
```

Use this method to write an integer value.

TWriter.Literal Method

```
procedure Literal(const Value: String)
```

Use this method to write a literal.

TWriter.NewLine Method

```
procedure NewLine
```

Use this method to write a new line (CRLF).

Note
New lines are not written if the JSON output is being compressed, or if the writer was created with the new line option turned off.

TWriter.NullProperty Method

```
procedure NullProperty(const Name: String)
```

Use this method to write a null property.

TWriter.NullValue Method

```
procedure NullValue
```

Use this method to write a null value.

TWriter.ObjectProperty Method

```
procedure ObjectProperty(const Name: String)
```

Use this method to write an object property's name using the PropertyName method.

TWriter.PropertyName Method

```
procedure PropertyName(const Name: String)
```

Use this method to write a property name.

TWriter.Separator Method

```
procedure Separator
```

Use this method to write a separator (,).

TWriter.StringProperty Method

```
procedure StringProperty(const Name: String; const Value:
      String)
```

Use this method to write a string property.

TWriter.StringValue Method

```
procedure StringValue(const Value: String)
```

Use this method to write a string value.

TWriter.Whitespace Method

```
procedure Whitespace
```

Use this method to write a space ( ).

Note
Whitespace is not written if the JSON output is being compressed.

10.230 TZoomControlOptions Component

Unit: WebMaps

Inherits From TMapOption

The TZoomControlOptions class controls how the zoom control is configured in a TMap control. These zoom control options correspond to the zoom control options available for maps in the Google Maps API.

Properties	Methods	Events
Position		
Style		

TZoomControlOptions.Position Property

```
property Position: TMapControlPosition
```

Specifies the position of the zoom control.

TZoomControlOptions.Style Property

```
property Style: TZoomControlStyle
```

Specifies the style of the zoom control.

Chapter 11
Type Reference

11.1 TAlertOrientation Type

Unit: WebLabels

```
TAlertOrientation = (aoLeft,aoRight)
```

The TAlertOrientation enumerated type is used with the TAlertLabel component to specify the orientation of the label's caption.

Element	Description
aoLeft	The label's caption will be positioned at the left side of the label.
aoRight	The label's caption will be positioned at the right side of the label.

11.2 TAnimatedIconDirection Type

Unit: WebIcons

```
TAnimatedIconDirection = (idVertical,idHorizontal)
```

The TAnimatedIconDirection enumerated type is used with the TAnimatedIcon component to specify the direction in which the frames of the animated icon are laid out.

Element	Description
idHorizontal	Specifies that the frames are laid out in a horizontal direction.
idVertical	Specifies that the frames are laid out in a vertical direction.

11.3 TAnimationCompleteEvent Type

Unit: WebCtrls

```
TAnimationCompleteEvent = procedure (Sender: TObject; Animation:
      TAnimation) of object
```

The TAnimationCompleteEvent type is a common event type that is used by controls to provide notification that an animation has completed for the control.

The Sender parameter represents the class instance that triggered the event. The Animation parameter represents the TAnimation instance that has completed.

11.4 TAnimationStyle Type

Unit: WebUI

```
TAnimationStyle = (asNone,asLinear,asQuadEaseOut,asQuadEaseIn,
      asQuadEaseInOut,asQuadEaseOutIn,asExpoEaseOut, asExpoEaseIn,
      asExpoEaseInOut,asExpoEaseOutIn, asCubicEaseOut,asCubicEaseIn,
      asCubicEaseInOut, asCubicEaseOutIn,asQuartEaseOut,asQuartEaseIn,
      asQuartEaseInOut,asQuartEaseOutIn,asQuintEaseOut, asQuintEaseIn,
      asQuintEaseInOut,asQuintEaseOutIn, asCircEaseOut,asCircEaseIn,
      asCircEaseInOut,asCircEaseOutIn, asSineEaseOut,asSineEaseIn,
      asSineEaseInOut,asSineEaseOutIn, asElasticEaseOut,
      asElasticEaseIn,asElasticEaseInOut, asElasticEaseOutIn,
      asBounceEaseOut,asBounceEaseIn, asBounceEaseInOut,
      asBounceEaseOutIn,asBackEaseOut, asBackEaseIn,asBackEaseInOut,
      asBackEaseOutIn)
```

The TAnimationStyle enumerated type is used to specify how an animation should transform a given property of a UI element or control.

Element	Description
asBackEaseIn	Easing equation function for a back easing in: accelerating from zero velocity.
asBackEaseInOut	Easing equation function for a back easing in/out: acceleration until halfway, then deceleration.
asBackEaseOut	Easing equation function for a back easing out: decelerating from zero velocity.
asBackEaseOutIn	Easing equation function for a back easing out/in: deceleration until halfway, then acceleration.
asBounceEaseIn	Easing equation function for a bounce (exponentially decaying parabolic bounce) easing in: accelerating from zero velocity.
asBounceEaseInOut	Easing equation function for a bounce (exponentially decaying parabolic bounce) easing in/out: acceleration until halfway, then deceleration.
asBounceEaseOut	Easing equation function for a bounce (exponentially decaying parabolic bounce) easing out: decelerating from zero velocity.
asBounceEaseOutIn	Easing equation function for a bounce (exponentially decaying parabolic bounce) easing out/in: deceleration until halfway, then acceleration.
asCircEaseIn	Easing equation function for a circular easing in: accelerating from zero velocity.

asCircEaseInOut	Easing equation function for a circular easing in/out: acceleration until halfway, then deceleration.
asCircEaseOut	Easing equation function for an exponential easing out/in: deceleration until halfway, then acceleration.
asCircEaseOutIn	Easing equation function for a circular easing in/out: acceleration until halfway, then deceleration.
asCubicEaseIn	Easing equation function for a cubic easing in: accelerating from zero velocity.
asCubicEaseInOut	Easing equation function for a cubic easing in/out: acceleration until halfway, then deceleration.
asCubicEaseOut	Easing equation function for a cubic easing out: decelerating from zero velocity.
asCubicEaseOutIn	Easing equation function for a cubic easing out/in: deceleration until halfway, then acceleration.
asElasticEaseIn	Easing equation function for an elastic (exponentially decaying sine wave) easing in: accelerating from zero velocity.
asElasticEaseInOut	Easing equation function for an elastic (exponentially decaying sine wave) easing in/out: acceleration until halfway, then deceleration.
asElasticEaseOut	Easing equation function for an elastic (exponentially decaying sine wave) easing out: decelerating from zero velocity.
asElasticEaseOutIn	Easing equation function for an elastic (exponentially decaying sine wave) easing out/in: deceleration until halfway, then acceleration.
asExpoEaseIn	Easing equation function for an exponential easing in: accelerating from zero velocity.
asExpoEaseInOut	Easing equation function for an exponential easing in/out: acceleration until halfway, then deceleration.
asExpoEaseOut	Easing equation function for an exponential easing out: decelerating from zero velocity.
asExpoEaseOutIn	Easing equation function for an exponential easing out/in: deceleration until halfway, then acceleration.
asLinear	Easing equation function for a simple linear tweening, with no easing.
asNone	No animation style.
asQuadEaseIn	Easing equation function for a quadratic easing in: accelerating from zero velocity.
asQuadEaseInOut	Easing equation function for a quadratic easing in/out: acceleration until halfway, then deceleration.
asQuadEaseOut	Easing equation function for a quadratic easing out: decelerating from zero velocity.

asQuadEaseOutIn	Easing equation function for a quadratic easing out/in: deceleration until halfway, then acceleration.
asQuartEaseIn	Easing equation function for a quartic easing in: accelerating from zero velocity.
asQuartEaseInOut	Easing equation function for a quartic easing in/out: acceleration until halfway, then deceleration.
asQuartEaseOut	Easing equation function for a quartic easing out: decelerating from zero velocity.
asQuartEaseOutIn	Easing equation function for a quartic easing out/in: deceleration until halfway, then acceleration.
asQuintEaseIn	Easing equation function for a quintic easing in: accelerating from zero velocity.
asQuintEaseInOut	Easing equation function for a quintic easing in/out: acceleration until halfway, then deceleration.
asQuintEaseOut	Easing equation function for a quintic easing out: decelerating from zero velocity.
asQuintEaseOutIn	Easing equation function for a quintic easing in/out: acceleration until halfway, then deceleration.
asSineEaseIn	Easing equation function for a sinusoidal easing in: accelerating from zero velocity.
asSineEaseInOut	Easing equation function for a sinusoidal easing in/out: acceleration until halfway, then deceleration.
asSineEaseOut	Easing equation function for a sinusoidal easing out: decelerating from zero velocity.
asSineEaseOutIn	Easing equation function for a sinusoidal easing in/out: deceleration until halfway, then acceleration.

11.5 TAuthenticationMethod Type

Unit: WebData

```
TAuthenticationMethod = (amHeaders,amParameters)
```

The TAuthenticationMethod enumerated type is used with the TDatabase component to specify how authentication should work for all dataset requests.

Element	Description

amHeaders	Specifies that HTTP headers will be used to pass authentication information to the web server for dataset requests. This is the default authentication method.
	The user name and password are added as custom headers to the web server request as follows:
	`X-EWBUser: <User Name>`
	`X-EWBPassword: <Password>`

amParameters	Specifies that URL parameters will be used to pass authentication information to the web server for dataset requests. The user name and password are added as parameters to the web server request as follows: `<DataSet Resource URL>&user=<User Name>&password=<Password>` **Warning** Elevate Web Builder uses the AJAX functionality in browsers to perform dataset requests, and this functionality is limited in its ability to perform authentication via native browser methods. Therefore, you should always use secure connections (https) to the web server with any dataset requests. This is especially true if using parameter-based authentication.

11.6 TAutoCompleteType Type

Unit: WebUI

```
TAutoCompleteType = (acDefault,acOn,acOff)
```

The TAutoCompleteType enumerated type is used with the descendant components of the TEditControl class to specify how auto-completion should be handled for the control. Auto-completion allows the browser to display a list of suggestions for input values, based upon earlier input values entered by the user.

Element	Description
acDefault	Specifies that auto-completion is enabled or disabled according to the default browser setting.
acOff	Specifies that auto-completion should be disabled.
acOn	Specifies that auto-completion should be enabled.

11.7 TBackgroundImageAnimateDirection Type

Unit: WebUI

```
TBackgroundImageAnimateDirection = (idVertical,idHorizontal)
```

The TBackgroundImageAnimateDirection enumerated type is used with the TBackgroundImage BeginAnimation method to specify in which direction the background image should be animated.

Note
The specified animation direction should match the actual orientation of the animation frames in the background image. If it doesn't match, then the animation will not appear correctly.

Element	Description
idHorizontal	Specifies that the background image should be animated in a horizontal direction.
idVertical	Specifies that the background image should be animated in a vertical direction.

11.8 TBackgroundImagePositionType Type

Unit: WebUI

```
TBackgroundImagePositionType = (ptSpecified,ptTopLeft,
      ptTopCenter,ptTopRight, ptCenterLeft,ptCenterCenter,
      ptCenterRight, ptBottomLeft,ptBottomCenter,ptBottomRight)
```

The TBackgroundImagePositionType enumerated type is used with the TBackgroundImage class to specify how background images should be positioned for UI elements and controls.

Element	Description
ptBottomCenter	Specifies that the background image should be centered horizontally at the bottom of the containing element or control.
ptBottomLeft	Specifies that the background image should be displayed at the bottom-left of the containing element or control.
ptBottomRight	Specifies that the background image should be displayed at the bottom-right of the containing element or control.
ptCenterCenter	Specifies that the background image should be centered horizontally and vertically within the containing element or control.
ptCenterLeft	Specifies that the background image should be centered vertically at the left of the containing element or control.
ptCenterRight	Specifies that the background image should be centered vertically at the right of the containing element or control.
ptSpecified	Specifies that the background image should be positioned according to the TBackgroundImage Left and Top properties.
ptTopCenter	Specifies that the background image should be centered horizontally at the top of the containing element or control.
ptTopLeft	Specifies that the background image should be displayed at the top-left of the containing element or control.
ptTopRight	Specifies that the background image should be displayed at the top-right of the containing element or control.

11.9 TBackgroundImageRepeatStyle Type

Unit: WebUI

```
TBackgroundImageRepeatStyle = (rsBoth,rsHorizontal,rsVertical,
      rsNone)
```

The TBackgroundImageRepeatStyle enumerated type is used with the TBackgroundImage class to specify how background images should be tiled, if at all, for UI elements and controls.

Note
The TBackgroundImage PositionType and SizeType properties, as well as the TBackground Origin and Clip properties, will affect how the background image is tiled.

Element	Description
rsBoth	Specifies that the background image should be tiled in both a horizontal and vertical direction.
rsHorizontal	Specifies that the background image should be tiled in a horizontal direction only.
rsNone	Specifies that the background image should not be tiled.
rsVertical	Specifies that the background image should be tiled in a vertical direction only.

11.10 TBackgroundImageSizeType Type

Unit: WebUI

```
TBackgroundImageSizeType = (stNone,stSpecified,stContain,
      stCover)
```

The TBackgroundImageSizeType enumerated type is used with the TBackgroundImage class to specify how background images should be sized for UI elements and controls.

Element	Description
stContain	Specifies that the background image should be proportionally sized so that it fits within the bounds of the element or control.
stCover	Specifies that the background image should be proportionally sized so that the image covers the bounds of the element or control.
stNone	Specifies that the background image should not be sized and should remain its original size.
stSpecified	Specifies that the background image should be sized according to the TBackgroundImage Width and Height properties.

11.11 TBackgroundOrientationType Type

Unit: WebUI

```
TBackgroundOrientationType = (otBounds,otBorder,otClient)
```

The TBackgroundOrientationType enumerated type is used with the TBackground class to specify how backgrounds should be positioned and clipped for UI elements and controls.

Element	Description
otBorder	Specifies that the background should originate and/or be clipped based upon the bounding rectangle of the UI element or control, minus any border.
otBounds	Specifies that the background should originate and/or be clipped based upon the bounding rectangle of the UI element or control.
otClient	Specifies that the background should originate and/or be clipped based upon the client rectangle of the border of the UI element or control. The client rectangle is the bounding rectangle minus any border or padding.

11.12 TBalloonOrientation Type

Unit: WebLabels

```
TBalloonOrientation = (boLeft,boCenter,boRight)
```

The TBalloonOrientation enumerated type is used with the TBalloonLabel component to specify the orientation of the balloon tail.

Element	Description
boCenter	The balloon tail will be positioned centered at the bottom of the label.
boLeft	The balloon tail will be positioned at the bottom-left corner of the label.
boRight	The balloon tail will be positioned at the bottom-right corner of the label.

11.13 TBooleanArray Type

Unit: WebCore

```
TBooleanArray = array of Boolean
```

The TBooleanArray type is used in classes such as the TSet class to represent an array of Boolean values.

11.14 TBorderStyle Type

Unit: WebUI

```
TBorderStyle = (bsSolid)
```

The TBorderStyle enumerated type is used with the TBorder class to specify the border style of one side of a border for UI elements and controls.

> **Note**
> This enumerated type currently has only one value. This is intentional, but will expand to include more options in a later minor release of Elevate Web Builder.

Element	Description
bsSolid	Specifies that the border should be a solid line.

11.15 TCalendarView Type

Unit: WebCals

```
TCalendarView = (cvMonth,cvYear,cvDecade,cvCentury)
```

The TCalendarView enumerated type is used with the descendant components of the TCalendarControl class to specify the active view for the calendar.

Element	Description
cvCentury	The calendar control will show a list of decades in the current century.
cvDecade	The calendar control will show a list of years in the current decade.
cvMonth	The calendar control will show a list of days in the current month (the default).
cvYear	The calendar control will show a list of months in the current year.

11.16 TCanPlayMedia Type

Unit: WebUI

```
TCanPlayMedia = (cpmCannot,cpmMaybe,cpmProbably)
```

The TCanPlayMedia enumerated type is used with the TMediaControl CanPlayMedia method and TMediaElement CanPlayMedia method to determine if a particular type of media can be played.

Element	Description
cpmCannot	The media can definitely not be played by the web browser.
cpmMaybe	The media may be able to be played by the web browser, but the browser is not certain.
cpmProbably	The media should be able to be played by the web browser.

11.17 TCanvasPoints Type

Unit: WebUI

```
TCanvasPoints = array of TCanvasPoint
```

The TCanvasPoints type is used in canvas drawing operations to represent an array of TCanvasPoint instances.

11.18 TCaption Type

Unit: WebLabels

```
TCaption = type String
```

The TCaption type is used to represent the caption of a control. This type is type-equivalent to a String type, but is used to distinguish the caption when used with special design-time property editors in order to allow for multi-line captions.

11.19 TCharArray Type

Unit: WebCore

```
TCharArray = array of Char
```

The TCharArray type is used in classes such as the TStringBuilder class to represent an array of Char values.

11.20 TClass Type

Unit: WebCore

```
TClass = class of TObject;
```

The TClass type is used to represent a class type, and allows TClass variables to store references to TObject classes and descendants.

11.21 TClickEvent Type

Unit: WebCtrls

```
TClickEvent = function (Sender: TObject): Boolean of object
```

The TClickEvent type is a common event type that is used by controls to provide notification that a sub-control has been clicked. For example, the TEditComboBox OnButtonClick event is used to allow the developer to intercept when the combo button is clicked in the control.

The Sender parameter represents the class instance that triggered the event.

To not allow the click, return False as the result to any event handler attached to this event. To allow the click, return True.

11.22 TCloseQueryEvent Type

Unit: WebCtrls

```
TCloseQueryEvent = function (Sender: TObject): Boolean of object
```

The TCloseQueryEvent type is used by the OnCloseQuery event for the TPanel, TForm, and TDialog components to intercept the closing of the control.

Return True as the result of the event to allow the close to continue, or False to prevent the control from closing.

11.23 TCollectionItemClass Type

Unit: WebCore

```
TCollectionItemClass = class of TCollectionItem
```

The TCollectionItemClass type is used to represent a TCollectionItem class type, and allows TCollectionItemClass variables to store references to TCollectionItem classes and descendants.

11.24 TCollectionItemName Type

Unit: WebCore

```
TCollectionItemName = type String
```

The TCollectionItemName type is used to represent the name of a TCollectionItem class instance. This type is type-equivalent to a String type, but is used to distinguish the name of a collection item instance when used with special design-time property editors.

11.25 TColor Type

Unit: WebUI

```
TColor = type Integer
```

The TColor type is an Integer type used to represent a 32-bit RGBA color. The components of the RGBA value are as follows:

Component	Description
Red	The blue component is stored in the lower 8 bits of the 32-bit RGBA value (0-7).
Green	The green component is stored in the next 8 bits of the 32-bit RGBA value (8-15).
Blue	The red component is stored in the next 8 bits of the 32-bit RGBA value (16-23).
Alpha	The alpha (opacity) component is stored in the last 8 bits of the 32-bit RGBA value (24-31).

11.26 TComponentClass Type

Unit: WebCore

```
TComponentClass = class of TComponent
```

The TComponentClass type is used to represent a TComponent class type, and allows TComponentClass variables to store references to TComponent classes and descendants.

11.27 TComponentName Type

Unit: WebCore

```
TComponentName = type String
```

The TComponentName type is used to represent the name of a TComponent class instance. This type is type-equivalent to a String type, but is used to distinguish the name of a component instance when used with special design-time property editors.

11.28 TCompositeOperation Type

Unit: WebUI

```
TCompositeOperation = (coSourceOver,coSourceOnTop,coSourceIn,
      coSourceOut, coDestOnTop,coDestIn,coDestOut,coDestOver,
      coLighter,coXOR,coCopy)
```

The TCompositeOperation enumerated type is used to specify the how pixels are combined when drawing occurs with a TCanvasElement instance.

Element	Description
coCopy	Draw the source pixel, ignoring the destination pixel.
coDestIn	Multiply the destination pixel by the opacity of the source pixel, but ignore the color of the source pixel.
coDestOnTop	Draw the source pixel underneath the destination pixel. If the source pixel is transparent, then the resulting pixel will also be transparent.
coDestOut	The destination pixel becomes transparent if the source pixel is opaque, and is not changed if the source pixel is transparent. The color of the source pixel is ignored.
coDestOver	The source pixel will appear behind the destination pixel, and shows based upon the transparency of the destination pixel.
coLighter	The colors of the source and destination pixels are added together and truncated by the maximum color value possible.
coSourceIn	Draw the source pixel, but multiply it by the opacity of the destination pixel. The color of the destination pixel is ignored, but if the destination pixel is transparent, then the resulting pixel will be also be transparent.
coSourceOnTop	Draw the source pixel on top of the destination, but multiply it by the opacity of the desination pixel. If the destination pixel is transparent, then nothing is drawn.
coSourceOut	The resulting pixel is the source pixel when the destination pixel is transparent, and a transparent pixel when the destination pixel is opaque. The color of the destination pixel is ignored.
coSourceOver	The source pixel is drawn on top of the destination pixel. If the source pixel is partially-transparent, then the destination pixel is included to draw the resulting pixel. This is the default type of composite operation for canvas drawing.
coXOR	If the source pixel is transparent, then the resulting pixel is the destination pixel. If the destination pixel is transparent, then the resulting pixel is the source pixel. If both the source and destination pixels are transparent or opaque, then the resulting pixel will be transparent.

11.29 TContent Type

Unit: WebLabels

```
TContent = type String
```

The TContent type is used to represent the content of a control. This type is type-equivalent to a String type, but is used to distinguish the content when used with special design-time property editors in order to allow for embedded HTML.

11.30 TContentAlignment Type

Unit: WebUI

```
TContentAlignment = (caLeft,caCenter,caRight)
```

The TContentAlignment enumerated type is used to specify the horizontal alignment of content in UI elements and controls.

Element	Description
caCenter	Specifies that the content will be centered.
caLeft	Specifies that the content will be left-justified.
caRight	Specifies that the content will be right-justified.

11.31 TContentDirection Type

Unit: WebUI

```
TContentDirection = (cdLeftToRight,cdRightToLeft)
```

The TContentDirection enumerated type is used to specify the text direction of content in UI elements and controls.

Element	Description
cdLeftToRight	Specifies that the content will be displayed/edited from left-to-right.
cdRightToLeft	Specifies that the content will be displayed/edited from right-to-left.

11.32 TContentPosition Type

Unit: WebCtrls

```
TContentPosition = (cpSpecified,cpTopLeft,cpTopCenter,cpTopRight,
       cpCenterLeft,cpCenterCenter,cpCenterRight, cpBottomLeft,
      cpBottomCenter,cpBottomRight)
```

The TContentPosition enumerated type is used with the TContentLayout class to specify how content elements should be positioned for UI elements and controls.

Element	Description
cpBottomCenter	Specifies that the content element should be centered horizontally at the bottom of the containing element or control.
cpBottomLeft	Specifies that the content element should be displayed at the bottom-left of the containing element or control.
cpBottomRight	Specifies that the content element should be displayed at the bottom-right of the containing element or control.
cpCenterCenter	Specifies that the content element should be centered horizontally and vertically within the containing element or control.
cpCenterLeft	Specifies that the content element should be centered vertically at the left of the containing element or control.
cpCenterRight	Specifies that the content element should be centered vertically at the right of the containing element or control.
cpSpecified	Specifies that the content element should be positioned according to the TContentLayout Left and Top properties.
cpTopCenter	Specifies that the content element should be centered horizontally at the top of the containing element or control.
cpTopLeft	Specifies that the content element should be displayed at the top-left of the containing element or control.
cpTopRight	Specifies that the content element should be displayed at the top-right of the containing element or control.

11.33 TContentSize Type

Unit: WebCtrls

```
TContentSize = (csNone,csSpecified,csContain,csCover)
```

The TContentSize enumerated type is used with the TContentLayout class to specify how content elements should be sized for UI elements and controls.

Element	Description
csContain	Specifies that the content element should be proportionally sized so that it fits within the bounds of the element or control.
csCover	Specifies that the content element should be proportionally sized so that the image covers the bounds of the element or control.
csNone	Specifies that the content element should not be sized and should remain its original size.
csSpecified	Specifies that the content element should be sized according to the TContentLayout Width and Height properties.

11.34 TControlClass Type

Unit: WebCtrls

```
TControlClass = class of TControl
```

The TControlClass type is used to represent a TControl class type, and allows TControlClass variables to store references to TControl classes and descendants.

11.35 TCursor Type

Unit: WebUI

```
TCursor = (crAuto,crCrossHair,crDefault,crHelp,crMove,crPointer,
      crProgress,crSizeNESW,crSizeNS,crSizeNWSE,crSizeWE, crText,
      crWait)
```

The TCursor enumerated type is used to specify the type of cursor that will be used for the mouse pointer when it hovers over a UI element or control.

Element	Description
crAuto	Specifies that the mouse pointer will use a cursor that is automatically determined by the web browser, based upon the type of control and its state.
crCrossHair	Specifies that the mouse pointer will use a cursor with a cross-hair (+) pattern.
crDefault	Specifies that the mouse pointer will use the default cursor for the web browser, which is usually a normal pointer.
crHelp	Specifies that the mouse pointer will use a help cursor, which is normally a pointer with a question mark (?) next to it.
crMove	Specifies that the mouse pointer will use a movement cursor, which is normally four arrows pointing north, east, south, and west.
crPointer	Specifies that the mouse pointer will use a normal pointer cursor.
crProgress	Specifies that the mouse pointer will use a cursor that represents execution in progress, which is normally a pointer with an animated progress symbol next to it.
crSizeNESW	Specifies that the mouse pointer will use a cursor that contains arrows pointing northeast and southwest.
crSizeNS	Specifies that the mouse pointer will use a cursor that contains arrows pointing north and south.
crSizeNWSE	Specifies that the mouse pointer will use a cursor that contains arrows pointing northwest and southeast. This is cursor is normally the same as the crSizeNESW cursor.
crSizeWE	Specifies that the mouse pointer will use a cursor that contains arrows pointing from west and east.
crText	Specifies that the mouse pointer will use a cursor that includes an edit caret (vertical bar).
crWait	Specifies that the mouse pointer will use a cursor that represents a wait state, which is normally an hourglass or animated progress symbol.

11.36 TDatabaseClass Type

Unit: WebData

```
TDatabaseClass = class of TDatabase
```

The TDatabaseClass type is used to represent a TDatabase class type, and allows TDatabaseClass variables to store references to TDatabase classes and descendants.

11.37 TDatabaseErrorEvent Type

Unit: WebData

```
TDatabaseErrorEvent = procedure (Sender: TObject; const
      ErrorMsg: String) of object
```

The TDatabaseErrorEvent type is used by the TDatabase OnCommitError and OnRollbackError events.

The Sender parameter is the TDatabase instance that triggered the event and the ErrorMsg parameter is the complete error message.

11.38 TDatabaseEvent Type

Unit: WebData

```
TDatabaseEvent = function (Sender: TObject): Boolean of object
```

The TDatabaseEvent type is used by the TDatabase BeforeCommit and BeforeRollback events.

The Sender parameter is the TDatabase instance that triggered the event. Return True from the event to allow the applicable database functionality to continue, or False to prevent the functionality from occurring.

11.39 TDataColumnTextEvent Type

Unit: WebData

```
TDataColumnTextEvent = function (Sender: TObject; const Value:
      String): String of object
```

The TDataColumnTextEvent type is used by the TDataColumn OnGetText and OnSetText events.

The Sender parameter is the TDataColumn instance that triggered the event and the Value parameter is the value of the column as a string (Get) or the display text being assigned to the column.

11.40 TDataRowEvent Type

Unit: WebData

```
TDataRowEvent = procedure (Sender: TObject; Column: TDataColumn)
      of object
```

The TDataRowEvent type is used by the TDataSet OnRowChanged event.

The Sender parameter is the TDataSet instance that triggered the event and the Column parameter is the column that was changed, if applicable. If the Column parameter is nil, then the entire row has been changed, as opposed to a single column.

11.41 TDataSetErrorEvent Type

Unit: WebData

```
TDataSetErrorEvent = procedure (Sender: TObject; const ErrorMsg:
      String) of object
```

The TDataSetErrorEvent type is used by the TDataSet OnLoadError event.

The Sender parameter is the TDataSet instance that triggered the event and the ErrorMsg parameter is the complete error message.

11.42 TDataSetEvent Type

Unit: WebData

```
TDataSetEvent = function (Sender: TObject): Boolean of object
```

The TDataSetEvent type is used by the TDataSet BeforeOpen, BeforeClose, BeforeScroll, BeforeInsert, BeforeUpdate, BeforeSave, BeforeCancel, BeforeDelete, and BeforeLoad events.

The Sender parameter is the TDataSet instance that triggered the event. Return True from the event to allow the applicable dataset functionality to continue, or False to prevent the functionality from occurring.

11.43 TDataSetState Type

Unit: WebData

```
TDataSetState = (dsClosed,dsBrowse,dsInsert,dsUpdate,dsFind,
      dsReset)
```

The TDataSetState enumerated type is used to indicate the State of a TDataSet instance.

Element	Description
dsBrowse	Indicates that the dataset is open and in the browse (read) state.
dsClosed	Indicates that the dataset is closed.
dsFind	Indicates that the dataset is open and in the find state. A dataset is put into the find state by calling the Find method.
dsInsert	Indicates that the dataset is open and in the insert state. A dataset is put into the insert state by calling the Insert method.
dsReset	Indicates that the dataset is being reset. This is an internal state used by the TDataSet component when it is loading or sorting rows.
dsUpdate	Indicates that the dataset is open and in the update state. A dataset is put into the update state by calling the Update method.

11.44 TDataType Type

Unit: WebCore

```
TDataType = (dtUnknown,dtString,dtBoolean,dtInteger,dtFloat,
      dtDate, dtTime,dtDateTime,dtBlob)
```

The TDataType enumerated type is used to specify the type of a TDataColumn instance in a TDataSet instance.

Element	Description
dtBlob	Specifies that the column is a BLOB (Binary Large Object) column.
dtBoolean	Specifies that the column is a boolean (True/False) column.
dtDate	Specifies that the column is a date column.
dtDateTime	Specifies that the column is a date/time column.
dtFloat	Specifies that the column is a floating-point precision numeric column.
dtInteger	Specifies that the column is an integer column.
dtString	Specifies that the column is a character string column.
dtTime	Specifies that the column is a time column.
dtUnknown	Specifies that the column is of an unknown type (do not use).

11.45 TDateTimeFormat Type

Unit: WebCore

```
TDateTimeFormat = (dtfRaw,dtfISO8601)
```

The TDateTimeFormat enumerated type is used with TReader and TWriter classes to specify how published DateTime properties are handled when reading/writing TPersistent-descendant classes to/from JSON.

Element	Description
dtfISO8601	Date-time values are handled as ISO-8601 date-time string values.
dtfRaw	Date-time values are handled as raw numeric DateTime values (the default).

11.46 TDrawStyle Type

Unit: WebUI

```
TDrawStyle = (dsColor,dsGradient,dsPattern)
```

The TDrawStyle enumerated type is used to specify the drawing style for stroke and fill operations on a TCanvasElement instance.

Element	Description
dsColor	Specifies that the drawing will use a solid color. This is the default drawing style.
dsGradient	Specifies that the drawing will use a gradient.
dsPattern	Specifies that the drawing will use a pattern.

11.47 TDropDownPosition Type

Unit: WebEdits

```
TDropDownPosition = (dpRelative,dpSurfaceCenter)
```

The TDropDownPosition enumerated type is used with the TEditComboBox, TDateEditComboBox, adn TButtonComboBox components to specify the position of the drop-down list or calendar.

Element	Description
dpRelative	Specifies that the drop-down control will be positioned relative to the owner control. This is usually beneath the owner control, but may also be above the control is there is insufficient space in which to show the entire drop-down control.
dpSurfaceCenter	Specifies that the drop-down control will be positioned in the center of the application surface. This is especially useful for mobile applications where the browser viewport is limited in size. The application surface is represented by the Surface property of the global Application variable in the WebForms unit.

11.48 TElementClass Type

Unit: WebUI

```
TElementClass = class of TElement
```

The TElementClass type is used to represent a TElement class type, and allows TElementClass variables to store references to TElement classes and descendants.

11.49 TErrorEvent Type

Unit: WebForms

```
TErrorEvent = function (Sender: TObject; const Message: String;
      const URL: String; Line: Integer): Boolean of object
```

The TErrorEvent type is used with the TApplication OnError event to trap unhandled exceptions that have bubbled up to the global Application object in a visual application.

Return True from the event to indicate to the web browser that the error was handled, or False to indicate to the web browser that the error should be handled by the browser.

11.50 TFillType Type

Unit: WebUI

```
TFillType = (ftSolid,ftGradient)
```

The TFillType enumerated type is used with the TFill class to specify how backgrounds should be filled for UI elements and controls.

Element	Description
ftGradient	Specifies that the background fill will be a gradient.
ftSolid	Specifies that the background fill will be a solid color (or transparent).

11.51 TFormControlClass Type

Unit: WebForms

```
TFormControlClass = class of TFormControl
```

The TFormControlClass type is used to represent a TFormControl class type, and allows TFormControlClass variables to store references to TFormControl classes and descendants.

11.52 TGenericFontFamily Type

Unit: WebUI

```
TGenericFontFamily = (gfSansSerif,gfSerif,gfMonospace,gfCursive,
      gfFantasy)
```

The TGenericFontFamily enumerated type is used with the TFont class to specify the generic font family for the fonts used in UI elements and controls. The generic font family is used as a fall-back if a specified font name is not available on the operating system that is hosting the web browser.

Note
At design-time, Elevate Web Builder will automatically set the generic font family when the font name is changed for a TFont class instance, so in most cases you will never need to modify the generic font family.

Element	Description
gfCursive	The cursive generic font family, which includes script fonts.
gfFantasy	The fantasy generic font family, which includes decorative fonts.
gfMonospace	The monospace generic font family, which includes all fixed-width fonts.
gfSansSerif	The sans-serif generic font family, which includes all fonts with plain stroke endings.
gfSerif	The serif generic font family, which includes all fonts with flared or decorative stroke endings.

11.53 TGradientType Type

Unit: WebUI

```
TGradientType = (gtLinear,gtRadial)
```

The TGradientType enumerated type is used with the TGradient class to specify the type of gradient background to be used for UI elements and controls.

Note
This setting is only valid if the UI element or control's background is set to use a gradient fill.

Element	Description
gtLinear	Specifies that the gradient will be linear. This is the default gradient type.
gtRadial	Specifies that the gradient will be radial.

11.54 TGridColumnCellEvent Type

Unit: WebGrids

```
TGridColumnCellEvent = procedure (Sender: TObject; ACell:
      TGridCell) of object
```

The TGridColumnCellEvent type is used by the TGridColumn OnCellUpdate event.

The Sender parameter is the TGridColumn instance that triggered the event and the ACell parameter is the TGridCell instance being updated.

11.55 TGridColumnCompareEvent Type

Unit: WebGrids

```
TGridColumnCompareEvent = function (const L,R: String;
      CaseInsensitive: Boolean; LocaleInsensitive: Boolean): Integer
      of object
```

The TGridColumnCompareEvent type is used by the TGridColumn OnCompare event.

The Sender parameter is the TGridColumn instance that triggered the event, the L and R parameters are the string column values to compare, and the CaseInsensitive and LocaleInsensitive parameters indicate what type of comparison is being requested.

11.56 TGridColumnControlType Type

Unit: WebGrids

```
TGridColumnControlType = (ctNone,ctEdit,ctEditComboBox,
      ctDialogEditComboBox, ctCheckBox,ctLink,ctIcon,ctImage,
      ctDateEditComboBox, ctMultiLineEdit,ctHTML)
```

The TGridColumnControlType enumerated type is used with the TGridColumn class to specify the type of control to be used when displaying or editing a grid column.

Element	Description
ctCheckBox	Specifies that the grid column will use a checkbox control for editing its cells.
ctDateEditComboBox	Specifies that the grid column will use a date combo box control for editing its cells.
ctDialogEditComboBox	Specifies that the grid column will use a dialog combo box control for editing its cells.
ctEdit	Specifies that the grid column will use a single-line edit control for editing its cells.
ctEditComboBox	Specifies that the grid column will use an edit combo box control for editing its cells.
ctHTML	Specifies that the grid column will display its cells as HTML.
ctIcon	Specifies that the grid column will display its cells as icons. The content of the cells specifies the name of the icon to display.
ctImage	Specifies that the grid column will display its cells as images. The content of the cells specifies the URL of the image to display.
ctLink	Specifies that the grid column will display its cells as links.
ctMultiLineEdit	Specifies that the grid column will use a multi-line edit control for editing its cells.
ctNone	Specifies that the grid column will not use any special controls for editing/displaying. This is the default value, and using this value will result in the grid column not being editable.

11.57 TGridHeaderClickEvent Type

Unit: WebGrids

```
TGridHeaderClickEvent = function (Sender: TObject): Boolean of
      object
```

The TGridHeaderClickEvent type is used by the TGridColumn OnHeaderClick event.

The Sender parameter is the TGridColumn instance that triggered the event.

11.58 THTMLFormEncoding Type

Unit: WebUI

```
THTMLFormEncoding = (feMultiPartFormData,feURLEncoded,
      feTextPlain)
```

The THTMLFormEncoding enumerated type is used with the THTMLForm and TFormElement classes to specify the type of encoding to use for any form values when the HTML form is submitted to the web server.

Element	Description
feMultiPartFormData	Specifies that the form should encode the form values using the multipart/form-data MIME type. This is the default type of encoding used in Elevate Web Builder, and should always be used when uploading files in addition to text values.
feTextPlain	Specifies that the form should encode the form values using the text/plain MIME type. The only encoding that will take place is that spaces will be converted into "+" characters.
feURLEncoded	Specifies that the form should encode the form values using the application/x-www-form-urlencoded MIME type. This encoding will cause spaces to be converted into "+" characters and any special characters to be converted into ASCII hex values prefixed with the "%" character.

11.59 THTMLFormMethod Type

Unit: WebUI

```
THTMLFormMethod = (fmGet,fmPost,fmHead,fmPut,fmDelete)
```

The THTMLFormMethod enumerated type is used to specify the HTTP method to use for the THTMLForm component when the Submit method is called to submit the form values to the web server.

Element	Description
fmDelete	Specifies that the HTTP DELETE method will be used to submit the HTML form values.
fmGet	Specifies that the HTTP GET method will be used to submit the HTML form values.
fmHead	Specifies that the HTTP HEAD method will be used to submit the HTML form values.
fmPost	Specifies that the HTTP POST method will be used to submit the HTML form values. This is the default method.
fmPut	Specifies that the HTTP PUT method will be used to submit the HTML form values.

11.60 TIntegerArray Type

Unit: WebCore

```
TIntegerArray = array of Integer
```

The TIntegerArray type is used in classes such as the TSet class to represent an array of Integer values.

11.61 TInterfaceAnimationEvent Type

Unit: WebUI

```
TInterfaceAnimationEvent = procedure (ACurrentTime: Double) of
      object
```

The TInterfaceAnimationEvent type is used by the TInterfaceManager BeginAnimation and ContinueAnimation methods to specify an animation event handler to be used for animating the frames of an animation.

The AStartTime parameter is the time, in milliseconds, when the animation frame was initiated. This value can be used by the event handler to reliably calculate animation effects based upon elapsed time.

11.62 TInterfaceControllerClass Type

Unit: WebUI

```
TInterfaceControllerClass = class of TInterfaceController
```

The TInterfaceControllerClass type is used to represent a TInterfaceController class type, and allows TInterfaceControllerClass variables to store references to TInterfaceController classes and descendants.

11.63 TInterfaceErrorEvent Type

Unit: WebUI

```
TInterfaceErrorEvent = function (const Message: String; const
      URL: String; Line: Integer): Boolean of object
```

The TInterfaceErrorEvent type is used by the TInterfaceManager class to specify a global error handler for an application.

> **Note**
> The global TApplication instance automatically created for visual applications assigns an event handler for interface manager errors, so you should not normally need to deal with this event type.

11.64 TInterfaceIdleEvent Type

Unit: WebUI

```
TInterfaceIdleEvent = procedure of object
```

The TInterfaceIdleEvent type is used by the TInterfaceManager OnIdle event to handle user inactivity timeouts.

11.65 TInterfaceTimeoutEvent Type

Unit: WebUI

```
TInterfaceTimeoutEvent = procedure of object
```

The TInterfaceTimeoutEvent type is used by the TInterfaceManager CreateTimeout method to create a timeout. A timeout waits N milliseconds, and then executes the specified event handler.

11.66 TInterfaceTimerEvent Type

Unit: WebUI

```
TInterfaceTimerEvent = procedure of object
```

The TInterfaceTimerEvent type is used by the TInterfaceManager CreateTimer method to create a timer. A timer executes the specified event handler every N milliseconds.

11.67 TInterfaceViewportResizeEvent Type

Unit: WebUI

```
TInterfaceViewportResizeEvent = procedure of object
```

The TInterfaceViewportResizeEvent type is used by the TInterfaceManager to indicate when the browser viewport has been resized.

11.68 TInterfaceViewportScrollEvent Type

Unit: WebUI

```
TInterfaceViewportScrollEvent = procedure of object
```

The TInterfaceViewportScrollEvent type is used by the TInterfaceManager to indicate when the browser viewport has been scrolled in any direction.

11.69 TKeyDownEvent Type

Unit: WebCtrls

```
TKeyDownEvent = function (Sender: TObject; Key: Integer;
      ShiftKey, CtrlKey, AltKey: Boolean): Boolean of object
```

The TKeyDownEvent type is a common event type that is used by controls to provide notification that a key is being pressed.

The Sender parameter represents the class instance that triggered the event. The Key parameter represents the ordinal key code of the key pressed, and the ShiftKey, CtrlKey, AltKey parameters represent whether the Shift, Control, and/or Alt keys were also pressed.

The Key value may represent a virtual key code. Certain keys, such as the Insert, Delete, and navigation keys, use virtual key codes since these keys do not correspond to an ordinal Unicode character value. The following virtual key code constants are defined in the WebUI unit for your convenience:

```
VK_BACK = 8;
VK_TAB = 9;
VK_RETURN = 13;
VK_ESCAPE = 27;
VK_SPACE = 32;
VK_PRIOR = 33;
VK_NEXT = 34;
VK_END = 35;
VK_HOME = 36;
VK_LEFT = 37;
VK_UP = 38;
VK_RIGHT = 39;
VK_DOWN = 40;
VK_INSERT = 45;
VK_DELETE = 46;
```

To not allow the keystroke, return False as the result to any event handler attached to this event. To allow the keystroke, return True.

11.70 TKeyPressEvent Type

Unit: WebCtrls

```
TKeyPressEvent = function (Sender: TObject; Key: Char; ShiftKey,
      CtrlKey, AltKey: Boolean): Boolean of object
```

The TKeyPressEvent type is a common event type that is used by controls to provide notification that a key has been pressed.

The Sender parameter represents the class instance that triggered the event. The Key parameter represents the character key code of the key pressed, and the ShiftKey, CtrlKey, AltKey parameters represent whether the Shift, Control, and/or Alt keys were also pressed.

Warning

Certain keys, such as the Insert, Delete, and navigation keys, use virtual key codes since these keys do not correspond to a Unicode character value. Only use this event to handle keystrokes that correspond to Unicode character values. Use the OnKeyDown event to handle keystrokes that use virtual key codes.

To not allow the keystroke, return False as the result to any event handler attached to this event. To allow the keystroke, return True.

11.71 TKeyUpEvent Type

Unit: WebCtrls

```
TKeyUpEvent = procedure (Sender: TObject; Key: Integer; ShiftKey,
       CtrlKey, AltKey: Boolean) of object
```

The TKeyUpEvent type is a common event type that is used by controls to provide notification that a key is being released.

The Sender parameter represents the class instance that triggered the event. The Key parameter represents the ordinal key code of the key pressed, and the ShiftKey, CtrlKey, AltKey parameters represent whether the Shift, Control, and/or Alt keys were also pressed.

The Key value may represent a virtual key code. Certain keys, such as the Insert, Delete, and navigation keys, use virtual key codes since these keys do not correspond to an ordinal Unicode character value. The following virtual key code constants are defined in the WebUI unit for your convenience:

```
VK_BACK = 8;
VK_TAB = 9;
VK_RETURN = 13;
VK_ESCAPE = 27;
VK_SPACE = 32;
VK_PRIOR = 33;
VK_NEXT = 34;
VK_END = 35;
VK_HOME = 36;
VK_LEFT = 37;
VK_UP = 38;
VK_RIGHT = 39;
VK_DOWN = 40;
VK_INSERT = 45;
VK_DELETE = 46;
```

11.72 TLayoutConsumption Type

Unit: WebUI

```
TLayoutConsumption = (lcNone,lcTopLeft,lcTop,lcTopRight,lcLeft,
      lcRight, lcBottomLeft,lcBottom,lcBottomRight)
```

The TLayoutConsumption enumerated type is used with the TLayout class to specify how a UI element or control consumes space in the layout rectangle.

Element	Description
lcBottom	Specifies that the element or control will consume space towards the bottom of the layout rectangle.
lcBottomLeft	Specifies that the element or control will consume space towards the bottom-left corner of the layout rectangle.
lcBottomRight	Specifies that the element or control will consume space towards the bottom-right corner of the layout rectangle.
lcLeft	Specifies that the element or control will consume space towards the left side of the layout rectangle.
lcNone	Specifies that the element or control will not consume any space in the layout rectangle. This is the default value.
lcRight	Specifies that the element or control will consume space towards the right side of the layout rectangle.
lcTop	Specifies that the element or control will consume space towards the top of the layout rectangle.
lcTopLeft	Specifies that the element or control will consume space towards the top-left corner of the layout rectangle.
lcTopRight	Specifies that the element or control will consume space towards the top-right corner of the layout rectangle.

11.73 TLayoutOverflow Type

Unit: WebUI

```
TLayoutOverflow = (loNone,loTop,loLeft,loRight,loBottom)
```

The TLayoutOverflow enumerated type is used with the TLayout class to specify how a UI element or control handles situations where it won't fit within the bounds of the current layout rectangle.

Element	Description
loBottom	Specifies that the prior element or control's consumption will be temporarily reset so that space is consumed towards the bottom of the current layout rectangle.
loLeft	Specifies that the prior element or control's consumption will be temporarily reset so that space is consumed towards the left side of the current layout rectangle.
loNone	Specifies that the prior element or control's consumption will not be temporarily reset.
loRight	Specifies that the prior element or control's consumption will be temporarily reset so that space is consumed towards the right side of the current layout rectangle.
loTop	Specifies that the prior element or control's consumption will be temporarily reset so that space is consumed towards the top of the current layout rectangle.

11.74 TLayoutPosition Type

Unit: WebUI

```
TLayoutPosition = (lpNone,lpTopLeft,lpTop,lpTopCenter,lpTopRight,
       lpLeft,lpLeftCenter,lpCenter,lpRight,lpRightCenter,
      lpBottomLeft,lpBottom,lpBottomCenter,lpBottomRight)
```

The TLayoutPosition enumerated type is used with the TLayout class to specify how a UI element or control is positioned in the layout rectangle.

Element	Description
lpBottom	Specifies that the element or control will be positioned at the bottom of the layout rectangle.
lpBottomCenter	Specifies that the element or control will be centered horizontally at the bottom of the layout rectangle.
lpBottomLeft	Specifies that the element or control will be positioned in the bottom left corner of the layout rectangle.
lpBottomRight	Specifies that the element or control will be positioned in the bottom right corner of the layout rectangle.
lpCenter	Specifies that the element or control will be centered horizontally and vertically within the layout rectangle.
lpLeft	Specifies that the element or control will be positioned on the left side of the layout rectangle.
lpLeftCenter	Specifies that the element or control will be centered vertically on the left side of the layout rectangle.
lpNone	Specifies that the element or control will not be positioned by the control's layout management, and will instead be positioned according to its assigned Left and Top property values. This is the default value.
lpRight	Specifies that the element or control will be positioned on the right side of the layout rectangle.
lpRightCenter	Specifies that the element or control will be centered vertically on the right side of the layout rectangle.
lpTop	Specifies that the element or control will be positioned at the top of the layout rectangle.
lpTopCenter	Specifies that the element or control will be centered horizontally at the top of the layout rectangle.
lpTopLeft	Specifies that the element or control will be positioned in the top left corner of the layout rectangle.
lpTopRight	Specifies that the element or control will be positioned in the top right corner of the layout rectangle.

11.75 TLayoutStretch Type

Unit: WebUI

```
TLayoutStretch = (lsNone,lsTopLeft,lsTop,lsTopRight,lsLeft,
      lsRight, lsBottomLeft,lsBottom,lsBottomRight)
```

The TLayoutStretch enumerated type is used with the TLayout class to specify how a UI element or control is stretched to fill parts of the layout rectangle.

Element	Description
lsBottom	Specifies that the element or control will stretch to the bottom of the layout rectangle.
lsBottomLeft	Specifies that the element or control will stretch to the bottom-left corner of the layout rectangle.
lsBottomRight	Specifies that the element or control will stretch to the bottom-right corner of the layout rectangle.
lsLeft	Specifies that the element or control will stretch to the left side of the layout rectangle.
lsNone	Specifies that the element or control will not be stretched. This is the default value.
lsRight	Specifies that the element or control will stretch to the right side of the layout rectangle.
lsTop	Specifies that the element or control will stretch to the top of the layout rectangle.
lsTopLeft	Specifies that the element or control will stretch to the top-left corner of the layout rectangle.
lsTopRight	Specifies that the element or control will stretch to the top-right corner of the layout rectangle.

11.76 TLineCapStyle Type

Unit: WebUI

```
TLineCapStyle = (csButt,csRound,csSquare)
```

The TLineCapStyle enumerated type is used to specify the how lines are terminated on a TCanvasElement instance.

Element	Description
csButt	Specifies that lines should have no cap. The end of the lines will be straight and perpendicular to the direction of the lines, and the lines are not extended beyond their endpoint.
csRound	Specifies that lines should be capped with a semicircle whose diameter is equal to the width of the line. This semicircle extends beyond the end of the line by one half of the width of the line.
csSquare	Specifies that lines should be capped with a rectangle. This is just like the csButt member, but the lines are extended by half of their width past their endpoint.

11.77 TLineJoinStyle Type

Unit: WebUI

```
TLineJoinStyle = (jsMiter,jsBevel,jsRound)
```

The TLineJoinStyle enumerated type is used to specify how lines are drawn when they intersect on a TCanvasElement instance.

Element	Description
jsBevel	Specifies that the outside edges of the intersecting lines are joined with a filled triangle.
jsMiter	Specifies that the outside edges of the intersecting lines are extended until they meet.
jsRound	Specifies that the outside edges of the intersecting lines are joined with a filled arc whose diameter is equal to the width of the line.

11.78 TLocationError Type

Unit: WebComps

```
TLocationError = (leNone,lePermissionDenied,leUnavailable,
      leTimeout)
```

The TLocationError enumerated type is used with the TLocationServices OnLocationError event to determine the reason why the current location cannot be obtained from the machine or device using the host web browser.

Element	Description
leNone	The location information hasn't been obtained yet, or was obtained without issue.
lePermissionDenied	The location information could not be obtained because the user did not grant permission for the current application to access the location information for the machine or device.
leTimeout	The location information could not be obtained for the machine or device within the number of milliseconds specified in the TLocationServices Timeout property.
leUnavailable	The location information could not be obtained because location services are not available for the machine or device. **Note** This error condition can occur when the current application was loaded in a non-secure context (http), while the host web browser only allows location services to be accessed from a secure context (https).

11.79 TMapControlPosition Type

Unit: WebMaps

```
TMapControlPosition = (cpBottomCenter,cpBottomLeft,cpBottomRight,
       cpLeftBottom,cpLeftCenter,cpLeftTop, cpRightBottom,
      cpRightCenter,cpRightTop, cpTopCenter,cpTopLeft,cpTopRight)
```

The TMapControlPosition enumerated type is used with the TMap control to specify how various map controls (map type, overview map, pan, rotate, street view, zoom) are positioned over the map.

Element	Description
cpBottomCenter	The control is positioned at the bottom of the map, in the center.
cpBottomLeft	The control is positioned at the bottom of the map, on the left.
cpBottomRight	The control is positioned at the bottom of the map, on the right.
cpLeftBottom	The control is positioned on the left side of the map, at the bottom.
cpLeftCenter	The control is positioned on the left side of the map, in the center.
cpLeftTop	The control is positioned on the left side of the map, at the top.
cpRightBottom	The control is positioned on the right side of the map, at the bottom.
cpRightCenter	The control is positioned on the right side of the map, in the center.
cpRightTop	The control is positioned on the right side of the map, at the top.
cpTopCenter	The control is positioned at the top of the map, in the center.
cpTopLeft	The control is positioned at the top of the map, on the left.
cpTopRight	The control is positioned at the top of the map, on the right.

11.80 TMapTilt Type

Unit: WebMaps

```
TMapTilt = (mt0Degrees,mt45Degrees)
```

The TMapTilt enumerated type is used with the TMapOptions class to specify the viewing angle of the map.

Element	Description
mt0Degrees	Specifies a viewing angle of 0 degrees (the default).
mt45Degrees	Specifies a viewing angle of 45 degrees.

11.81 TMapType Type

Unit: WebMaps

```
TMapType = (mtHybrid,mtRoadmap,mtSatellite,mtTerrain)
```

The TMapType enumerated type is used with the TMapOptions class to specify the type of map to show.

Element	Description
mtHybrid	This map type displays a transparent layer of major streets on satellite images.
mtRoadmap	This map type displays a normal street map.
mtSatellite	This map type displays satellite images.
mtTerrain	This map type displays maps with physical features such as terrain and vegetation.

11.82 TMapTypeControlStyle Type

Unit: WebMaps

```
TMapTypeControlStyle = (tcDefault,tcDropDownMenu,
      tcHorizontalBar)
```

The TMapTypeControlStyle enumerated type is used with the TMapTypeControlOptions class to specify the style of the map type control.

Element	Description
tcDefault	The map type control will be the default control style (a horizontal menu bar).
tcDropDownMenu	The map type control will be a drop-down menu.
tcHorizontalBar	The map type control will be a horizontal menu bar.

11.83 TMediaNetworkState Type

Unit: WebUI

```
TMediaNetworkState = (mnsEmpty,mnsIdle,mnsLoading,mnsNoSource)
```

The TMediaNetworkState enumerated type is used with the TAudio, TVideo, and TMediaElement classes to determine the current network state of a media UI element or control.

Element	Description
mnsEmpty	The media element or control has not started using the network. This is the case right after a new media URL has been specified for the media element or control.
mnsIdle	The media element or control is not currently using the network.
mnsLoading	The media element or control is currently using the network to load media data.
mnsNoSource	The media element or control is not currently using the network because the media URL specified as a source cannot be played by the media element or control.

11.84 TMediaPreload Type

Unit: WebUI

```
TMediaPreload = (mplNone,mplMetaData,mplAuto)
```

The TMediaPreload enumerated type is used with the TAudio, TVideo, and TMediaElement classes to specify how much media data can be pre-loaded by the media UI element or control.

Element	Description
mplAuto	All of the media data can be pre-loaded.
mplMetaData	Only the media metadata can be pre-loaded.
mplNone	None of the media data can be pre-loaded.

11.85 TMediaReadyState Type

Unit: WebUI

```
TMediaReadyState = (mrsNothing,mrsMetadata,mrsCurrentData,
      mrsFutureData,mrsEnoughData)
```

The TMediaReadyState enumerated type is used with the TAudio, TVideo, and TMediaElement classes to determine the current playback-readiness of the media control.

Element	Description
mrsCurrentData	Media data has been loaded for the current playback position, but nothing more. This state normally occurs at the end of media playback.
mrsEnoughData	Enough media data has been loaded that the media control should be able to play until the end of the media without pausing.
mrsFutureData	Enough media data has been loaded to begin playing, but the media control will most likely have to pause at some later point to load more data.
mrsMetadata	The media metadata has been loaded, so statistics such as the duration of the media will be available, but no media data has been loaded yet.
mrsNothing	No media data has been loaded (including metadata).

11.86 TModalResult Type

Unit: WebCtrls

```
TModalResult = (mrNone,mrOk,mrCancel,mrAbort,mrRetry, mrIgnore,
      mrYes,mrNo,mrAll,mrNoToAll, mrYesToAll,mrClose)
```

The TModalResult enumerated type is used with modal forms and dialogs to indicate which action the user selected to close the modal form or dialog.

Element	Description
mrAbort	Indicates that the Abort button was clicked or selected.
mrAll	Indicates that the All button was clicked or selected.
mrCancel	Indicates that the Cancel button was clicked or selected.
mrClose	Indicates that the Close button was clicked or selected.
mrIgnore	Indicates that the Ignore button was clicked or selected.
mrNo	Indicates that the No button was clicked or selected.
mrNone	Indicates that no button was clicked or selected.
mrNoToAll	Indicates that the No to All button was clicked or selected.
mrOk	Indicates that the Ok button was clicked or selected.
mrRetry	Indicates that the Retry button was clicked or selected.
mrYes	Indicates that the Yes button was clicked or selected.
mrYesToAll	Indicates that the Yes to All button was clicked or selected.

11.87 TMouseDownEvent Type

Unit: WebCtrls

```
TMouseDownEvent = procedure (Sender: TObject; Button: Integer;
      ShiftKey, CtrlKey, AltKey: Boolean; X,Y: Integer) of object
```

The TMouseDownEvent type is a common event type that is used by controls to provide notification that a mouse button is being pressed.

The Sender parameter represents the class instance that triggered the event. The Button parameter represents the ordinal button code of the mouse button pressed, and the ShiftKey, CtrlKey, AltKey parameters represent whether the Shift, Control, and/or Alt keys were also pressed. The X and Y parameters indicate the horizontal and vertical position of the mouse pointer, in pixels, relative to the bounds of the control that triggered the event.

The ordinal mouse button values are defined in the WebUI unit, and are as follows:

```
MB_NONE = 0;
MB_LEFT = 1;
MB_MIDDLE = 2;
MB_RIGHT = 3;
```

11.88 TMouseMoveEvent Type

Unit: WebCtrls

```
TMouseMoveEvent = procedure (Sender: TObject; ShiftKey, CtrlKey,
      AltKey: Boolean; X,Y: Integer) of object
```

The TMouseMoveEvent type is a common event type that is used by controls to provide notification that the mouse pointer is being moved over the control.

The Sender parameter represents the class instance that triggered the event. The ShiftKey, CtrlKey, AltKey parameters represent whether the Shift, Control, and/or Alt keys were also pressed. The X and Y parameters indicate the horizontal and vertical position of the mouse pointer, in pixels, relative to the bounds of the control that triggered the event.

The ordinal mouse button values are defined in the WebUI unit, and are as follows:

```
MB_NONE = 0;
MB_LEFT = 1;
MB_MIDDLE = 2;
MB_RIGHT = 3;
```

11.89 TMouseUpEvent Type

Unit: WebCtrls

```
TMouseUpEvent = procedure (Sender: TObject; Button: Integer;
      ShiftKey, CtrlKey, AltKey: Boolean; X,Y: Integer) of object
```

The TMouseUpEvent type is a common event type that is used by controls to provide notification that a mouse button is being released.

The Sender parameter represents the class instance that triggered the event. The Button parameter represents the ordinal button code of the mouse button pressed, and the ShiftKey, CtrlKey, AltKey parameters represent whether the Shift, Control, and/or Alt keys were also pressed. The X and Y parameters indicate the horizontal and vertical position of the mouse pointer, in pixels, relative to the bounds of the control that triggered the event.

The ordinal mouse button values are defined in the WebUI unit, and are as follows:

```
MB_NONE = 0;
MB_LEFT = 1;
MB_MIDDLE = 2;
MB_RIGHT = 3;
```

11.90 TMouseWheelEvent Type

Unit: WebCtrls

```
TMouseWheelEvent = procedure (Sender: TObject; WheelDelta:
      Integer; ShiftKey, CtrlKey, AltKey: Boolean; X,Y: Integer) of
      object
```

The TMouseWheelEvent type is a common event type that is used by controls to provide notification that the mouse wheel is being rotated.

The Sender parameter represents the class instance that triggered the event. The WheelDelta parameter represents the amount, in pixels, that the mouse wheel rotation represents, and the ShiftKey, CtrlKey, AltKey parameters represent whether the Shift, Control, and/or Alt keys were also pressed. The X and Y parameters indicate the horizontal and vertical position of the mouse pointer, in pixels, relative to the bounds of the control that triggered the event.

Note
The WheelDelta parameter can be positive or negative, depending upon the configuration of the mouse and the direction in which the mouse wheel was rotated.

11.91 TMsgDlgBtn Type

Unit: WebForms

```
TMsgDlgBtn = (mbNone,mbOk,mbCancel,mbAbort,mbRetry,mbIgnore,
      mbYes,mbNo,mbAll,mbNoToAll,mbYesToAll,mbClose)
```

The TMsgDlgBtn enumerated type is used to specify the message dialog buttons to display in the MessageDlg procedure.

Element	Description
mbAbort	Specifies that the button is an Abort button.
mbAll	Specifies that the button is an All button.
mbCancel	Specifies that the button is a Cancel button.
mbClose	Specifies that the button is a Close button.
mbIgnore	Specifies that the button is an Ignore button.
mbNo	Specifies that the button is a No button.
mbNone	Not used.
mbNoToAll	Specifies that the button is a No to All button.
mbOk	Specifies that the button is an Ok button.
mbRetry	Specifies that the button is a Retry button.
mbYes	Specifies that the button is a Yes button.
mbYesToAll	Specifies that the button is a Yes to All button.

11.92 TMsgDlgBtns Type

Unit: WebForms

```
TMsgDlgBtns = array of TMsgDlgBtn
```

The TMsgDlgBtns type is simply a type definition for an array of TMsgDlgBtn enumerated values, and is used in the MessageDlg procedure.

11.93 TMsgDlgResultEvent Type

Unit: WebForms

```
TMsgDlgResultEvent = procedure (DlgResult: TModalResult) of
      object
```

The TMsgDlgResultEvent event type is used with the MessageDlg procedure to pass an event handler to the procedure that is called when the modal message dialog is closed by the user.

The DlgResult parameter indicates the button that the user clicked or selected, if any, to close the message dialog.

11.94 TMsgDlgType Type

Unit: WebForms

```
TMsgDlgType = (mtWarning,mtError,mtInformation,mtConfirmation,
      mtCustom)
```

The TMsgDlgType enumerated type is used with the MessageDlg procedure to specify what type of message dialog should be displayed.

Element	Description
mtConfirmation	Specifies that the message dialog will be a confirmation dialog, and an applicable icon will be displayed on the message dialog to reflect this.
mtCustom	Specifies that the message dialog will be a custom dialog, and an applicable icon will be displayed on the message dialog to reflect this.
mtError	Specifies that the message dialog will be an error dialog, and an applicable icon will be displayed on the message dialog to reflect this.
mtInformation	Specifies that the message dialog will be an informational dialog, and an applicable icon will be displayed on the message dialog to reflect this.
mtWarning	Specifies that the message dialog will be a warning dialog, and an applicable icon will be displayed on the message dialog to reflect this.

11.95 TNotifyEvent Type

Unit: WebCore

```
TNotifyEvent = procedure (Sender: TObject) of object
```

The TNotifyEvent type is a common event type that is used in any situation where a simple notification mechanism is required, such as the OnClick event. The Sender parameter is the class instance that triggered the event.

11.96 TObjectsArray Type

Unit: WebCore

```
TObjectsArray = array of TObject
```

The TObjectsArray type is used in classes like the TObjectList class to represent an array of TObject instances.

11.97 TOverflowType Type

Unit: WebUI

```
TOverflowType = (otHidden,otAuto,otScroll)
```

The TOverflowType type is used by the TViewport component to specify whether or not horizontal and/or vertical scrollbars should be shown when the application surface size exceeds the browser viewport size.

Element	Description
otAuto	A scrollbar is shown if the application surface exceeds the browser viewport size, otherwise no scrollbar is shown.
otHidden	No scrollbar is shown, even if the application surface exceeds the browser viewport size This is the default value.
otScroll	A scrollbar is always shown, even if the application surface does not exceed the browser viewport size.

11.98 TPageChangeEvent Type

Unit: WebPages

```
TPageChangeEvent = function (Sender: TObject; NewPage: TPage):
      Boolean of object
```

The TPageChangeEvent event type is used by the TPagePanel OnPageChange event to indicate when the active page changes in the control.

11.99 TPatternRepeatStyle Type

Unit: WebUI

```
TPatternRepeatStyle = (psNone,psHorizontal,psVertical,psBoth)
```

The TPatternRepeatStyle enumerated type is used to specify how a pattern should be tiled on a TCanvasElement instance.

Element	Description
psBoth	Specifies that the pattern will be tiled both horizontally and vertically.
psHorizontal	Specifies that the pattern will be tiled horizontally.
psNone	Specifies that the pattern will not be tiled at all. This is the default value.
psVertical	Specifies that the pattern will be tiled vertically.

11.100 TRequestMethod Type

Unit: WebHTTP

```
TRequestMethod = (rmGet,rmPost,rmHead,rmPut,rmDelete,rmPatch)
```

The TRequestMethod enumerated type is used with the TServerRequest component to specify the HTTP method for a web server request.

Element	Description
rmDelete	Specifies that the request is an HTTP DELETE request.
rmGet	Specifies that the request is an HTTP GET request.
rmHead	Specifies that the request is an HTTP HEAD request.
rmPatch	Specifies that the request is an HTTP PATCH request. **Note** This method is not supported by the Elevate Web Builder Web Server and is only provided here for usage with other web servers.
rmPost	Specifies that the request is an HTTP POST request.
rmPut	Specifies that the request is an HTTP PUT request.

11.101 TScrollBars Type

Unit: WebCtrls

```
TScrollBars = (sbNone,sbVertical,sbHorizontal,sbBoth)
```

The TScrollBars enumerated type is used with various scrollable controls to specify how scrollbars should be displayed when their content overflows the client area of the control.

Element	Description
sbBoth	Specifies that both a horizontal and vertical scrollbar should be displayed, if necessary.
sbHorizontal	Specifies that a horizontal scrollbar should be displayed, if necessary.
sbNone	Specifies that no scrollbars should be displayed.
sbVertical	Specifies that a vertical scrollbar should be displayed, if necessary.

11.102 TScrollSupport Type

Unit: WebCtrls

```
TScrollSupport = (ssNone,ssVertical,ssHorizontal,ssBoth)
```

The TScrollSupport enumerated type is used with various scrollable controls to specify the directions in which the controls can be scrolled when their content overflows the client area of the control.

> **Note**
> This property only applies to scrolling via touch and mouse wheel movements, and does not apply to programmatic scrolling or scrolling via the scroll bars themselves.

Element	Description
ssBoth	Specifies that scrolling is allowed in both the horizontal and vertical directions.
ssHorizontal	Specifies that scrolling is allowed in the horizontal direction only.
ssNone	Specifies that scrolling is not allowed in either the horizontal or vertical direction.
ssVertical	Specifies that scrolling is allowed in the vertical direction only.

11.103 TSelectionState Type

Unit: WebBtns

```
TSelectionState = (ssIndeterminate,ssUnselected,ssSelected)
```

The TSelectionState enumerated type is used with the TCheckBox and TRadioButton classes to specify the selection state of the control.

Element	Description
ssIndeterminate	Specifies that no selection has been made. This is the default value.
ssSelected	Specifies that the control is selected.
ssUnselected	Specifies that the control is not selected.

11.104 TServerRequestEvent Type

Unit: WebHTTP

```
TServerRequestEvent = procedure (Request: TServerRequest) of
      object
```

The TServerRequestEvent event type is used by the TServerRequest OnComplete event to indicate when a server request is complete.

The Request parameter indicates the server request that triggered the event, and the StatusCode property of the server request can be examined to determine if the request completed successfully.

11.105 TServerRequestProgressEvent Type

Unit: WebHTTP

```
TServerRequestProgressEvent = procedure (Current: Integer;
      Total: Integer) of object
```

11.106 TServerRequestURL Type

Unit: WebHTTP

```
TServerRequestURL = type String
```

The TServerRequestURL type is used to represent the URL of a TServerRequest class instance. This type is type-equivalent to a String type, but is used to distinguish the URL of a server request instance when used with special design-time property editors.

11.107 TSizerOrientation Type

Unit: WebSizer

```
TSizerOrientation = (soVertical,soHorizontal)
```

The TSizerOrientation enumerated type is used with the TSizer control to specify in which direction a sizer control be oriented, which determines the direction in which the associated control will be resized as the sizer control is moved.

Element	Description
soHorizontal	The sizer control will size a control in a horizontal direction.
soVertical	The sizer control will size a control in a vertical direction.

11.108 TSlideEvent Type

Unit: WebSlide

```
TSlideEvent = procedure (Sender: TObject; SlideIndex: Integer;
      const SlideImageURL: String) of object
```

The TSlideEvent type is used by the OnLoadSlide and OnRenderSlide events for the TSlideShow control to intercept the loading or rendering of slide images.

11.109 TSortDirection Type

Unit: WebData

```
TSortDirection = (sdNone,sdAscending,sdDescending)
```

The TSortDirection enumerated type is used with the TDataColumn class to specify how a column should be sorted in a dataset.

Element	Description
sdAscending	Specifies that the column should be sorted in ascending order.
sdDescending	Specifies that the column should be sorted in descending order.
sdNone	Specifies that the column should not be part of the active sort.

11.110 TStorageChangeEvent Type

Unit: WebComps

```
TStorageChangeEvent = procedure (Sender: TObject; const Key:
      String; const NewValue: String; const OldValue: String; const
      URL: String) of object
```

The TStorageChangeEvent event type is used by the TPersistentStorage OnChange event to indicate when the persistent local storage (but not the session-only local storage) is changed by another session in the host web browser.

The Key parameter indicates the key of the item that was updated. This parameter will be blank if the contents of the persistent local storage were removed using the ClearAll method.

The NewValue parameter indicates the value of the item that was updated. This parameter will be blank if the contents of the persistent local storage were removed using the ClearAll method, or if the item was removed using the Clear method.

const Key: String;
const NewValue: String; const OldValue: String;
const URL: String

The Request parameter indicates the server request that triggered the event, and the StatusCode property of the server request can be examined to determine if the request completed successfully.

11.111 TStringsArray Type

Unit: WebCore

```
TStringsArray = array of String
```

The TStringArray type is used in classes like the TStringList class to represent an array of String values.

11.112 TTextAlignment Type

Unit: WebUI

```
TTextAlignment = (taLeftJustify,taCenter,taRightJustify)
```

The TTextAlignment enumerated type is used to specify the horizontal alignment of text on a TCanvasElement instance.

Element	Description
taCenter	Specifies that the text will be centered.
taLeftJustify	Specifies that the text will be left-justified.
taRightJustify	Specifies that the text will be right-justified.

11.113 TTextBaseLine Type

Unit: WebUI

```
TTextBaseLine = (blAlphabetic,blTop,blMiddle,blBottom,blHanging,
      blIdeographic)
```

The TTextBaseLine enumerated type is used to specify the vertical alignment of text on a TCanvasElement instance.

Element	Description
blAlphabetic	Specifies that the baseline is the normal alphabetic baseline. This is the default vertical alignment.
blBottom	Specifies that the baseline is the bottom of the bounding box that encompasses the text.
blHanging	Specifies that the baseline is the hanging baseline.
blIdeographic	Specifies that the baseline is the ideographic baseline.
blMiddle	Specifies that the baseline is the middle of the em square.
blTop	Specifies that the baseline is the top of the em square.

11.114 TTextInputType Type

Unit: WebUI

```
TTextInputType = (tiNone,tiEmail,tiNumber,tiURL)
```

The TTextInputType enumerated type is used with the TTextInputElement class to specify how the text should be input. This information is used with touch interfaces to determine the type of soft keyboard to display when inputting text into the element.

Element	Description
tiEmail	Specifies that the element will contain an email address.
tiNone	Specifies that the element will contain regular text.
tiNumber	Specifies that the element will contain a number.
tiURL	Specifies that the element will contain a URL.

11.115 TTouchEvent Type

Unit: WebCtrls

```
TTouchEvent = procedure (Sender: TObject; ShiftKey, CtrlKey,
      AltKey: Boolean; X,Y: Integer) of object
```

The TTouchEvent type is a common event type that is used by controls to provide notification that a touch event is occurring via a touch interface.

The Sender parameter represents the class instance that triggered the event. The ShiftKey, CtrlKey, AltKey parameters represent whether the Shift, Control, and/or Alt keys were also pressed. The X and Y parameters indicate the horizontal and vertical position of the touch, in pixels, relative to the bounds of the control that triggered the event.

11.116 TTouchScrollEvent Type

Unit: WebCtrls

```
TTouchScrollEvent = procedure (Sender: TObject; X,Y: Integer) of
      object
```

The TTouchScrollEvent type is a common event type that is used by controls to provide notification that a control is being scrolled using a touch movement in any direction.

The Sender parameter represents the class instance that triggered the event. The X parameter indicates the horizontal movement, in pixels, while the Y parameter indicates the vertical movement, in pixels.

11.117 TWebElementEvent Type

Unit: WebUI

```
TWebElementEvent = procedure (AElement: TWebElement) of object
```

The TWebElementEvent type is used with the TWebElement OnLoad, OnUnload, and OnError events.

11.118 TZoomControlStyle Type

Unit: WebMaps

```
TZoomControlStyle = (zcDefault,zcLarge,zcSmall)
```

The TZoomControlStyle enumerated type is used with the TZoomControlOptions class to specify the size of the zoom control.

Element	Description
zcDefault	The default size (large).
zcLarge	A large zoom control.
zcSmall	A small zoom control.