



TMS WEB Core

v1.6.2.0 Pesaro
DEVELOPERS GUIDE

Index

Introduction	23
Scope and architecture	23
Additional resources	24
Online videos.....	24
Online training course.....	24
Books	25
Getting started	28
Configuring TMS WEB Core settings	34
Configuring TMS WEB Core project settings.....	36
Automatic versioning.....	37
Installation for Lazarus.....	39
Your first TMS WEB Core application	41
Your first TMS WEB Core progressive web application.....	45
Your first TMS WEB Core Electron Application	50
Debugging	52
Pascal to JavaScript Compiler	55
RTL.....	56
Preprocessor	57
Command-line compiler	58
Utility functions.....	61
Page Design	64
Absolute positioning.....	64
Relative positioning.....	65
Theming.....	66
BiDiMode	68
Use of HTML templates	69
JavaScript and CSS.....	75
Using off the shelf HTML templates	76
Automatic synchronisation with HTML templates	92
Live preview.....	94
Working with databases	97
Solutions with REST APIs for classic databases	98
Solutions with REST APIs for cloud database solutions	99
Existing REST APIs.....	100
Application	101
Forms	106
Creating forms at runtime	108
Hosting forms in other controls	110
Form inheritance	112

Frames	113
UI control types	113
UI controls encapsulating HTML elements	114
Custom drawn controls using the HTML5 CANVAS element	114
TMS FNC controls	114
jQuery UI controls	115
Standard Components	116
Common properties of visual controls	116
Common events of visual controls	118
TWebLabel	119
Description	119
HTML template tag	119
Properties for TWebLabel	120
Events for TWebLabel	121
TWebButton	121
Description	121
HTML template tag	121
Properties for TWebButton	122
Events for TWebButton	122
TWebEdit	123
Description	123
HTML template tag	123
Properties for TWebEdit	123
Methods for TWebEdit	124
Events for TWebEdit	125
TWebEditAutocomplete	125
Description	125
HTML template tag	126
Properties for TWebEditAutocomplete	126
Events for TWebEditAutocomplete	127
TWebSpinEdit	127
Description	127
HTML template tag	128
Properties for TWebSpinEdit	128
Events for TWebSpinEdit	129
TWebMaskEdit	129
Description	129
Properties for TWebMaskEdit	130
Events for TWebMaskEdit	130
TWebDateTimePicker	131
Description	131
HTML template tag	131

Properties for TWebDateTimePicker	131
Events for TWebDateTimePicker	132
TWebListBox	133
Description	133
HTML template tag	133
Properties for TWebListBox	133
Events for TWebListBox	134
TWebComboBox	135
Description	135
HTML template tag	135
Properties for TWebComboBox	135
Events for TWebComboBox	136
TWebLookupComboBox	136
Description	136
HTML template tag	137
Properties for TWebLookupComboBox	137
Events for TWebLookupComboBox	138
TWebColorPicker	138
Description	138
HTML template tag	139
Properties for TWebColorPicker	139
Events for TWebColorPicker	139
TWebCheckBox	140
Description	140
Properties for TWebCheckBox	140
Events for TWebCheckBox	141
TWebRadioButton	141
Description	141
Properties for TWebRadioButton	141
Events for TWebRadioButton	142
TWebMainMenu	142
Description	142
HTML template tag	143
Properties for TWebMainMenu	143
Events for TWebMainMenu	145
TWebMemo	145
Description	145
HTML template tag	146
Properties for TWebMemo	146
Methods for TWebMemo	147
Events for TWebMemo	147
TWebRadioGroup	147

Description	147
Properties for TWebRadioGroup	147
Events for TWebRadioGroup	148
TWebCheckGroup	148
Description	148
Properties for TWebCheckGroup	149
Events for TWebCheckGroup	149
TWebProgressBar	150
Description	150
HTML template tag	150
Properties for TWebProgressBar	150
TWebBadge	151
Description	151
HTML template tag	151
Properties for TWebBadge	151
TWebPaintBox	152
Description	152
Properties for TWebPaintBox	152
Events for TWebPaintBox	152
TWebTrackBar	154
Description	154
HTML template tag	154
Properties for TWebTrackBar	154
Events for TWebTrackBar	155
TWebScrollBar	155
Description	155
HTML template tag	156
Properties for TWebScrollBar	156
Events for TWebScrollBar	156
TWebSplitter	157
Description	157
Properties for TWebSplitter	157
Events for TWebSplitter	158
TWebSignIn	158
Description	158
HTML template tag	158
Properties for TWebSignIn	158
Methods for TWebSignIn	159
Events for TWebSignIn	159
TWebPanel	160
Description	160
HTML template tag	160

Properties for TWebPanel	161
Events for TWebPanel	161
TWebHTMLContainer	161
Description	161
HTML template tag	162
Properties for TWebHTMLContainer	162
Events for TWebHTMLContainer	163
TWebHTMLForm	163
Description	163
HTML template tag	164
Properties for TWebHTMLForm	164
TWebHTMLDiv	165
Description	165
HTML template tag	165
Properties for TWebHTMLDiv	165
TWebHTMLSpan	166
Description	166
HTML template tag	166
Properties for TWebHTMLSpan	166
TWebHTMLAnchor	167
Description	167
HTML template tag	167
Properties for TWebHTMLAnchor	167
TWebImageControl	168
Description	168
HTML template tag	168
Properties for TWebImageControl	168
Methods for TWebImageControl	169
Events for TWebImageControl	169
TWebImageZoomControl	170
Description	170
HTML template tag	170
Properties for TWebImageControl	170
Events for TWebImageControl	171
TWebLinkLabel	172
Description	172
HTML template tag	172
Properties for TWebLinkLabel	173
Events for TWebLinkLabel	173
TWebRichEdit	174
Description	174
HTML template tag	174

Properties for TWebRichEdit	174
Events for TWebRichEdit	175
TWebSyntaxMemo	175
Loading a file	176
Downloading a file.....	177
Properties for TWebSyntaxMemo	177
Methods for TWebSyntaxMemo	178
Events for TWebSyntaxMemo.....	180
TWebTabSet	181
Description	181
HTML template tag	181
Properties for TWebTabSet	181
Methods for TWebTabSet	182
Events for TWebTabSet.....	182
TWebPageControl	182
Description	182
HTML template tag	183
Properties for TWebPageControl	183
Methods for TWebPageControl	184
Events for TWebPageControl.....	184
TWebTabsheet.....	184
Description	185
TWebLoginPanel	186
Description	186
Properties for TWebLoginPanel	186
Events for TWebLoginPanel.....	187
TWebSpeedButton	188
Description	188
HTML template tag	188
Properties for TWebSpeedButton	188
Events for TWebSpeedButton.....	189
TWebPayPal	190
Description	190
HTML template tag	190
Properties for TWebPayPal.....	190
Events for TWebPayPal	192
TWebToolbar.....	194
Description	194
Properties for TWebToolbar	194
Events for TWebToolbar	194
TWebRichEditToolbar	196
Description	196

Properties for TWebRichEditToolbar	196
Events for TWebRichEditToolbar	197
TWebGridPanel	197
Description	197
HTML template tag	197
Properties for TWebGridPanel	198
Events for TWebGridPanel	199
TWebTreeview	199
Description	199
HTML template tag	199
Properties for TWebTreeview	200
Methods for TWebTreeview	200
Events for TWebTreeview	200
Sample code	201
TWebAccordion	202
Description	202
HTML template tag	202
Properties for TWebAccordion	202
Events for TWebAccordion	203
Properties for TAccordionSection	203
TWebResponsiveGridPanel	204
Description	204
HTML template tag	204
Properties for TWebResponsiveGridPanel	204
Methods for TWebResponsiveGridPanel	205
Events for TWebResponsiveGridPanel	205
Properties for TResponsiveLayoutItem	205
TWebMessageDlg	206
Description	206
Properties for TWebMessageDlg	207
Methods for TWebMessageDlg	207
Events for TWebMessageDlg	207
TWebWaitMessage	208
Description	208
Properties for TWebWaitMessage	208
Methods for TWebWaitMessage	208
Events for TWebWaitMessage	209
TWebFileUpload	210
Description	210
Properties for TWebFileUpload	210
HTML template tag	210
Events for TWebFileUpload	210

Properties for TFile	211
Methods for TFile	212
TWebFilePicker	214
Description	214
HTML template tag	214
Properties for TWebFilePicker	214
Events for TWebFilePicker	214
Example code	215
TWebShare	216
Description	216
Methods for TWebShare	216
TWebOpenDialog	217
Description	217
Properties for TWebOpenDialog	217
Methods for TWebOpenDialog	217
Events for TWebOpenDialog	217
TWebToast	218
Description	218
Properties for TWebToast	218
Events for TWebToast	219
Properties for TToastItem	219
Methods for TToastItem	219
TWebToggleButton	220
Description	220
Properties for TWebToggleButton	220
Events for TWebToggleButton	220
TWebBitBtn	221
Description	221
HTML template tag	221
Properties for TWebBitBtn	222
Events for TWebBitBtn	223
TWebGroupBox	224
Description	224
HTML template tag	224
Properties for TWebGroupBox	224
Events for TWebGroupBox	225
TWebStretchPanel	226
Description	226
HTML template tag	227
Properties for TWebStretchPanel	228
Events for TWebStretchPanel	228
TWebStringGrid	229

Description	229
HTML template tag	229
Properties for TWebStringGrid	230
Methods for TWebStringGrid	230
Events for TWebStringGrid	231
TWebListControl	233
Description	233
Properties for TListItem	234
Methods for TListItem	234
Properties for TWebListControl	234
Events for TWebListControl	235
TWebTableControl	236
Description	236
HTML template tag	237
Properties for TWebTableControl	237
Methods for TWebTableControl	237
Events for TWebTableControl	238
TWebResponsiveGrid	240
Description	240
HTML template tag	240
Properties for TWebResponsiveGrid	241
Methods for TWebResponsiveGrid	242
Events for TWebResponsiveGrid	242
Properties for TWebResponsiveGridItem	244
TWebLoginPanel	245
Description	245
Properties for TWebLoginPanel	245
Events for TWebLoginPanel	246
TWebImageSlider	247
Description	247
Properties for TWebImageSlider	247
Public properties for TWebImageSlider	248
Methods for TWebImageSlider	248
Events for TWebImageSlider	249
Example code	249
TWebContinuousScroll	250
Description	250
Properties for TWebContinuousScroll	251
Methods for TWebContinuousScroll	252
Events for TWebContinuousScroll	252
TWebSignatureCapture	255
Description	255

Properties for TWebSignatureCapture	255
Methods for TWebSignatureCapture	255
TWebCalendar	256
Properties for TWebCalendar	256
Methods for TWebCalendar	257
Events for TWebCalendar	257
TWebGoogleReCaptcha	258
Description	258
Properties for TWebGoogleReCaptcha	258
Methods for TWebGoogleReCaptcha	258
Events for TWebGoogleReCaptcha	258
TWebGoogleDrive	259
Description	259
HTML template tag	259
Properties for TWebGoogleDrive	259
TWebGoogleMaps	260
Description	260
HTML template tag	260
Properties for TWebGoogleMaps	261
Methods for TWebGoogleMaps	262
Events for TWebGoogleMaps	265
TWebGoogleChart	266
Description	266
Properties for TWebGoogleChart	268
Methods for TWebGoogleChart	270
Events for TWebGoogleChart	270
Examples	271
TWebSentry	274
Steps to set up	274
First bring up Project Settings	275
Continuing with the rest of the Demo	282
Properties for TWebSentry	291
Methods for TWebSentry	291
TWebBrowserControl	293
Description	293
HTML template tag	293
Properties for TWebBrowserControl	294
TWebMultimediaPlayer	294
Description	294
HTML template tag	295
Properties for TWebMultimediaPlayer	295
TWebMediaCapture	297

Description	297
Properties for TWebMediaCapture	297
Methods for TWebMediaCapture	298
Events for TWebMediaCapture	298
TWebYoutube	299
Description	299
HTML template tag	299
Properties for TWebYoutube	299
TWebTwitterFeed	301
Description	301
HTML template tag	302
Properties for TWebTwitterFeed	302
TWebCamera	302
Description	302
Selecting a device	303
Starting the camera stream automatically	304
HTML template tag	304
Properties for TwebCamera	304
Methods for TWebCamera	305
Events for TWebCamera	305
DB-aware components	307
TWebDataSource	307
Description	307
Properties for TWebDataSource	308
TWebClientDataSet	308
Description	308
Properties for TWebClientDataSet	308
Methods for TWebClientDataSet	309
Events for TWebClientDataSet	310
TWebClientConnection	311
Description	311
Properties for TWebClientConnection	311
Events for TWebClientConnection	312
TWebDBLabel	313
Description	313
TWebDBEdit	313
Description	313
TWebDBEditAutoComplete	313
Description	313
TWebDBCheckBox	314
Description	314
TWebDBSpinEdit	314

Description	314
TWebDBMaskEdit	314
Description	314
TWebDBComboBox	314
Description	315
TWebDBLookupComboBox	315
Description	315
TWebDBMemo	315
Description	315
TWebDBDateTimePicker	315
Description	316
TWebDBRadioGroup	316
Description	316
TWebDBLinkLabel	316
Description	316
TWebDBImageControl	316
Description	317
TWebDBTableControl	318
TWebDBResponsiveGrid	318
Description	318
TWebDBGrid	319
Description	319
TWebDBNavigator	320
Description	320
Non-visual components and classes	321
TWebTimer	321
TWebClipboard	321
TWebBluetooth	321
TWebBluetooth class	322
TWebBluetoothDevice class	322
TWebBluetoothService class	323
TWebBluetoothCharacteristic class	324
TWebElementActionList	326
Description	326
Properties for TWebElementActionList	327
Events for TWebElementActionList	327
Properties for TWebElementAction	328
Methods for TWebElementAction	329
Events for TWebElementAction	330
Example	330
TWebLocalStorage	331
TWebSessionStorage	331

TWebURLValidator.....	332
Properties for TWebURLValidator	332
Events for TWebURLValidator	332
TWebGeoLocation.....	333
Methods for TWebGEOLocation	333
Events for TWebGEOLocation	333
TWebSocketClient.....	334
Properties for TWebSocketClient	334
Methods for TWebSocketClient.....	335
Events for TWebSocketClient	335
TWebHttpRequest.....	335
Properties for TWebHttpRequest	338
Methods for TWebHttpRequest.....	338
Events for TWebHttpRequest.....	339
TWebCookies.....	340
TWebClientConnector	341
TWebAESEncryption.....	344
Properties for TWebAESEncryption	344
Methods for TWebAESEncryption.....	344
Events for TWebAESEncryption	345
TWebRSAEncryption.....	345
Properties for TWebRSAEncryption	345
Methods for TWebRSAEncryption	346
Events for TWebRSAEncryption	347
TWebRSASignature	347
Properties for TWebRSASignature.....	348
Methods for TWebRSASignature	348
Events for TWebRSASignature	349
TWebHMACSignature	349
Properties for TWebHMACSignature	349
Methods for TWebHMACSignature	350
Events for TWebHMACSignature.....	350
TWebPushNotifications	350
Registration for push notifications	351
Multiple users on the same device	351
Properties for TWebPushNotifications.....	352
Methods for TWebPushNotifications	352
Events for TWebPushNotifications.....	353
TMS WEB Core 3D	354
Your first 3D Chart application	354
3D Business Chart Applications.....	356
The 3D Bar Chart Demo	356

The Terminology for Axes	357
Creating the Data Series object	357
Other features shown in the Demo	358
Events	359
3D Math Chart Applications	359
The 3D Scatter Chart Demo	359
The Terminology for Axes	359
Creating the Data Series object	360
Other features shown in the Demo	360
The 3D Surface Chart Demo	360
How it works	361
Creating the Data Series object	361
Other features shown in the Demo	362
3D PaintBox Applications	362
The 3D PaintBox Demo	363
The Terminology for Axes	363
The code for adding objects	363
Direct Use of the Three.Js API	364
Sample code for Other features	364
Events	365
3D Model Applications	365
“ThreeJS Models (3d)” JS Library is required	365
The 3D Model Demo	365
The Code to Load OBJ/MTL Models	366
The code to Load GLTF Model	366
Additional features for Models	367
TWebMyCloudDbClientDataset Component	368
Introduction	368
Your first web application using TWebmyCloudDbClientDataset	368
Set up your myCloudData project in the myCloudData console	368
Create a TMS Web Application	368
Set up the TWebmyCloudDbClientDataset component	368
Specify the Component Properties	369
Create the Fields or Properties of each object in the Object Store	369
Select the fields in the Object Inspector	369
Create the Fields in code	369
Add Data Components that connect to the DataSet	369
Set up the DataSource and Data components	370
Set up the Columns of the DBGrid	370
Set up a New Record event	370
Run the Web Application	370

Todo List Demo	370
Additional features in this Demo.....	370
Troubleshooting	371
Reference Section.....	371
TWebMyCloudDbClientDataset	371
Properties of TWebmyCloudDbClientDataSet.....	371
Methods of TWebmyCloudDbClientDataset.....	372
TWebFirestoreClientDataset Component.....	374
Introduction	374
Your first web application using TWebFirestoreClientDataset.....	374
Set up your Firestore project in the Firebase console	374
Create a TMS web application	375
Add Data Components that connect to the DataSet	377
Run the Web Application.....	377
Todo List Demo	378
Troubleshooting	378
TWebFirestoreClientDataset reference.....	378
Methods of TWebFirestoreClientDataset	379
TWebRadServerClientDataset	381
Introduction	381
Configuring your Embarcadero Rad server back-end	381
Use Rad Server via TWebRadServerClientDataset.....	384
Reference.....	384
Properties	384
Methods.....	385
Events.....	385
TWebDreamFactoryClientDataset.....	386
Introduction	386
Configuring your DreamFactory back-end	386
Create the SQLite Service 'tasksdb'	386
Create Schema Table Task	386
Setup CORS	388
Set up a Role "LoggedIn" to access the "tasksdb" service	388
Create App Tasks	388
Using DreamFactory via TWebDreamFactoryClientDataset	389
Reference.....	389
Properties	390
Methods.....	390
Events.....	390
TWebFaunaDbClientDataSet.....	391
Introduction	391
Configuring the FaunaDb server back-end	391

Using FaunaDB via TWebFaunaDBClientDataset	393
Reference.....	394
Properties	394
Methods.....	394
Events.....	395
TWebSQLRestClientDataset, TWebSQLRestConnection	396
Introduction	396
Configuring the SQLDBRESTBridge server back-end	396
Using SQLite via TWebSQLRestClientDataset.....	397
jQuery components.....	400
TWebJQXButton	402
Description.....	402
HTML template tag	402
Properties for TWebJQXButton.....	402
Events for TWebJQXButton	403
TWebJQXButtonGroup.....	404
Description.....	404
HTML template tag	404
Properties for TWebJQXButtonGroup.....	404
Events for TWebJQXButtonGroup	405
TWebJQXCalendar	406
Description.....	406
HTML template tag	406
Properties for TWebJQXCalendar.....	406
Events for TWebJQXCalendar	407
TWebJQXColorPicker	408
Description.....	408
HTML template tag	408
Properties for TWebJQXColorPicker.....	408
Events for TWebJQXColorPicker	409
TWebJQXComboBox	410
Description.....	410
HTML template tag	410
Properties for TWebJQXComboBox.....	410
Methods for TWebJQXComboBox	411
Events for TWebJQXComboBox.....	411
TWebJQXDateTimeInput.....	412
Description.....	412
HTML template tag	412
Properties for TWebJQXDateTimeInput.....	412
Events for TWebJQXDateTimeInput	413
TWebJQXDropDownList	414

Description	414
HTML template tag	414
Properties for TWebJQXDropDownList	414
Methods for TWebJQXDropDownList	415
Events for TWebJQXDropDownList	415
TWebJQXGrid	416
Description	416
HTML template tag	417
Properties for TWebJQXGrid	417
Methods for TWebJQXGrid	419
Events for TWebJQXGrid	419
TWebJQXKnob	421
Description	421
HTML template tag	421
Properties for TWebJQXKnob	421
Events for TWebJQXKnob	423
TWebJQXMaskedInput	424
Description	424
HTML template tag	424
Properties for TWebJQXMaskedInput	424
Events for TWebJQXMaskedInput	425
TWebJQXMenu	426
Description	426
HTML template tag	426
Properties for TWebJQXMenu	426
Events for TWebJQXMenu	427
TWebJQXNumberInput	428
Description	428
HTML template tag	428
Properties for TWebJQXNumberInput	428
Events for TWebJQXNumberInput	429
TWebJQXProgressBar	430
Description	430
HTML template tag	430
Properties for TWebJQXProgressBar	430
TWebJQXRangeSelector	432
Description	432
HTML template tag	432
Properties for TWebJQXRangeSelector	432
Events for TWebJQXRangeSelector	433
TWebJQXRating	434
Description	434

HTML template tag	434
Properties for TWebJQXRating	434
Events for TWebJQXRating	435
TWebJQXResponsivePanel	435
Description	435
Properties for TWebJQXResponsivePanel	436
Methods for TWebJQXResponsivePanel	436
Events for TWebJQXResponsivePanel	436
TWebJQXSlider	437
Description	437
HTML template tag	437
Properties for TWebJQXSlider	437
Events for TWebJQXSlider	438
TWebJQXTabs	439
Description	439
HTML template tag	439
Properties for TWebJQXTabs	439
Events for TWebJQXTabs	440
TWebJQXTagCloud	440
Description	440
HTML template tag	441
Properties for TWebJQXTagCloud	441
Events for TWebJQXTagCloud	442
Connecting to data	443
Using WebSockets	448
IndexedDB	453
TMS WEB Core IndexedDB Library	453
TIndexedDbClientDataSet Component	453
TIndexedDb class	453
Your first IndexedDB application	453
Create a TMS web application	453
Add DB-aware components that connect to the DataSet	456
Run the Web Application	456
Managing the IndexedDB Database	456
Todo List Demo	457
Additional features in this Demo	457
TWebIndexedDbClientDataSet	458
Description	458
Properties of TWebIndexedDbClientDataSet	458
Methods of TWebIndexedDbClientDataSet	459
TIndexedDb (Advanced Use)	460
Description	460

Properties	460
Methods and Events of TIndexedDb	461
<i>Out-of-line key specification</i>	463
Setting up the Indexes	467
Handling Asynchronous behavior of IndexedDB	468
TMS WEB Electron	470
Your first TMS Web Electron Application	470
Building the application.....	473
Migrate your application to newer versions.....	474
Replace the main.js file	474
Dialog callbacks	474
Remove the Sender parameter from TElectronIPCCommunication.OnMessage.....	475
Remove TElectronIPCMain related codes.....	475
Accessing the Developer Tools	476
Drag and drop	476
Fonts	478
Set up your project with local databases.....	479
Electron Components.....	480
TElectronOpenDialog	480
Description.....	480
Properties for TElectronOpenDialog.....	480
Methods for TElectronOpenDialog	481
TElectronSaveDialog.....	481
Description.....	481
Properties for TElectronSaveDialog	481
Methods for TElectronSaveDialog.....	482
TElectronMessageBox	482
Description.....	482
Properties for TElectronMessageBox.....	482
Methods for TElectronMessageBox	483
TElectronErrorBox.....	483
Description.....	483
Properties for TElectronErrorBox	484
Methods for TElectronErrorBox.....	484
TElectronMainMenu	484
Description.....	484
TElectronPopupMenu.....	485
Description.....	485
Methods for TElectronPopupMenu.....	485
TElectronBrowserWindow	485
Description.....	485

Properties for TElectronBrowserWindow	486
Methods for TElectronBrowserWindow	486
Events for TElectronBrowserWindow	487
Multiple windows using forms.....	487
Multiple windows using other sources	488
Showing a window	488
TElectronTrayIcon	488
Description	488
Properties for TElectronTrayIcon	489
Events for TElectronTrayIcon.....	489
TElectronIPCCommunication	489
Description	489
Properties for TElectronIPCCommunication	489
Methods for TElectronIPCCommunication	489
Events for TElectronIPCCommunication	489
Send message to parent.....	490
Send message to a channel.....	490
Receiving messages.....	490
TElectronMySQLClientDataSet	491
Description	491
Todo List Demo.....	491
BLOB demo	492
Properties for TElectronMySQLClientDataSet.....	492
TElectronMySQLConnection	493
Description	493
TElectronPostgreSQLClientDataSet.....	493
Description	493
Todo List Demo.....	493
BLOB demo	494
Properties for TElectronPostgreSQLClientDataSet	495
TElectronPostgreSQLConnection.....	495
Description	495
TElectronFileWatcher	495
Description	495
Properties for TElectronFileWatcher	495
TElectronFileWatch.....	495
TElectronGlobalShortcut	496
Description	496
Properties for TElectronGlobalShortcut.....	496
Events for TElectronGlobalShortcut	496
TElectronStringList.....	496
Methods for TElectronStringList	496

TElectronBinaryDataStream	498
Properties for TElectronBinaryDataStream	498
Methods for TElectronBinaryDataStream	498
TElectronClipboard	498
Properties of TElectronClipboard	498
Methods of TElectronClipboard	498
TElectronShell	499
Methods for TElectronShell	499
TElectronIPCRenderer	499
Methods for TElectronIPCRenderer	499
TElectronDragAndDrop	500
TElectronPath	501
Methods for TElectronPath	501
TElectronWindow	501
Methods for TElectronWindow	501
Other available methods	502
Custom control development	503
Appendix	514
Browser locale values	514
TUILanguage	516

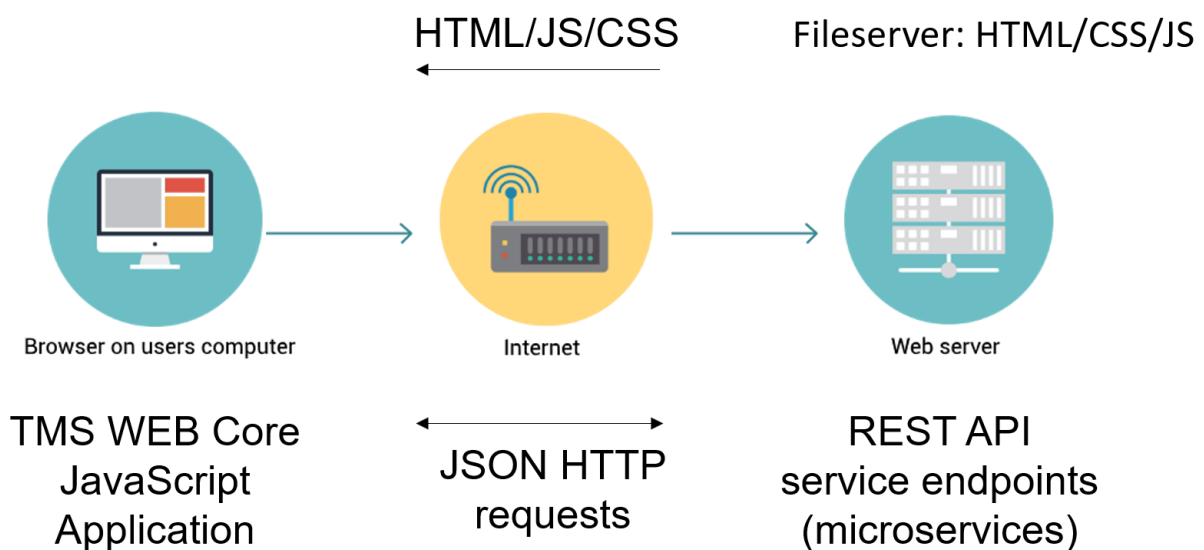
Introduction

Welcome to TMS WEB Core. TMS WEB Core is the foundation of an exciting new and modern way for creating web client applications from Delphi using RAD development methodology and using a component-based framework.

Scope and architecture

TMS WEB Core is based on compiling Delphi UI code to JavaScript and creating this way what is called Single-Page-Applications. The terminology “page” should not be confused with forms that Delphi developers are used to. A TMS WEB Core application can contain multiple forms. These multiple forms will be hosted in a JavaScript application a web browser user can navigate to via a single page URL. Any modern HTML5 compliant browser can run TMS WEB Core web client applications. This includes Chrome, Safari, Edge, Firefox, Firefox Developer Edition, Opera.

We will further refer to TMS WEB Core applications as web client applications. This means applications running as JavaScript code in the browser client (left). The web client application will typically communicate with a server or servers for working with data or other services (right). The TMS WEB Core web client application is open to work with different server technologies. This includes but is not limited to TMS XData, Embarcadero RAD Server, node.js, ASP.NET Core microservices. The typical technology used for this communication is via HTTP REST APIs.



More information about using TMS XData as a backend for TMS WEB Core web client applications can be found at:

<https://download.tmssoftware.com/business/xdata/doc/web/web-applications-tms-web-core.html>

TMS XData provides 3 components to make it easier to consume a TMS XData REST API:

TXDataWebConnection:

<https://download.tmssoftware.com/business/xdata/doc/web/txdatawebconnection.html>

TXDataWebClient:

<https://download.tmssoftware.com/business/xdata/doc/web/using-txdatawebclient.html>

TXDataWebDataset:

<https://download.tmssoftware.com/business/xdata/doc/web/using-txdatawebdataset.html>

Additional resources

In addition to this product manual and the various sample applications included in the product are additional resources

Online videos

We have produced several videos explaining specific functionality in the TMS WEB Core framework:

<https://www.tmssoftware.com/site/videos.asp?EN=on&DE=on&PT=on&vcatsel=9>

Online training course

Landgraf.dev is offering an extensive online video course explaining TMS WEB Core, its architecture, its components, working with templates and connecting with databases in the backend:

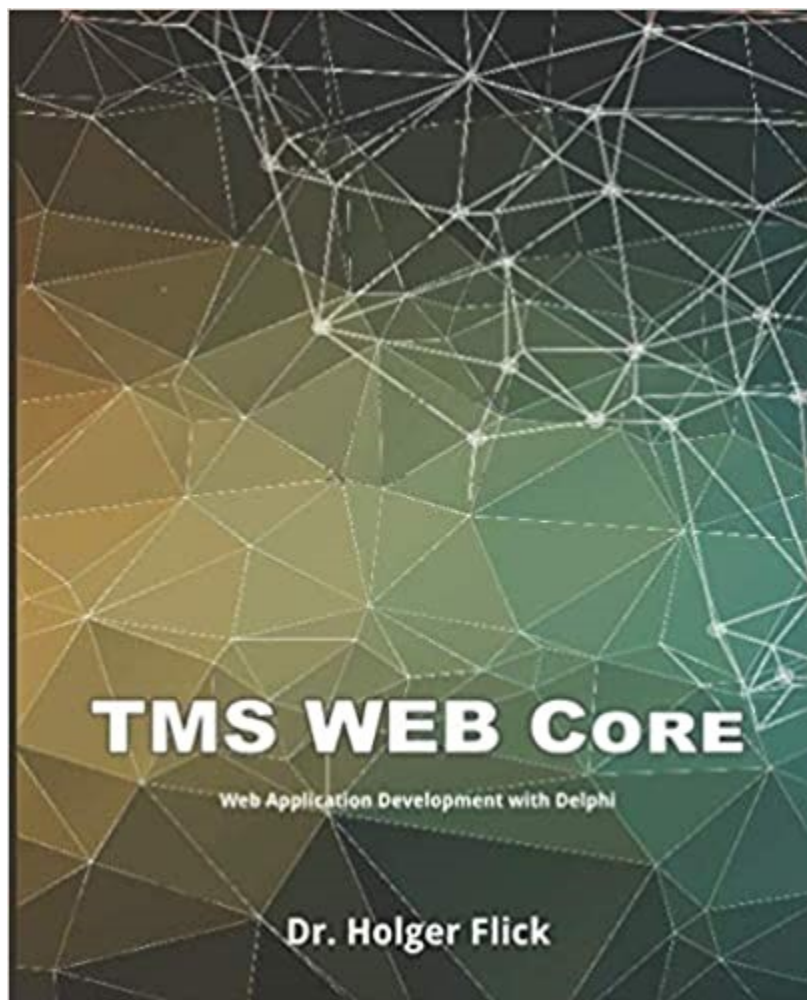
<https://courses.landgraf.dev/p/web-applications-with-delphi-tms-web-core>

Books

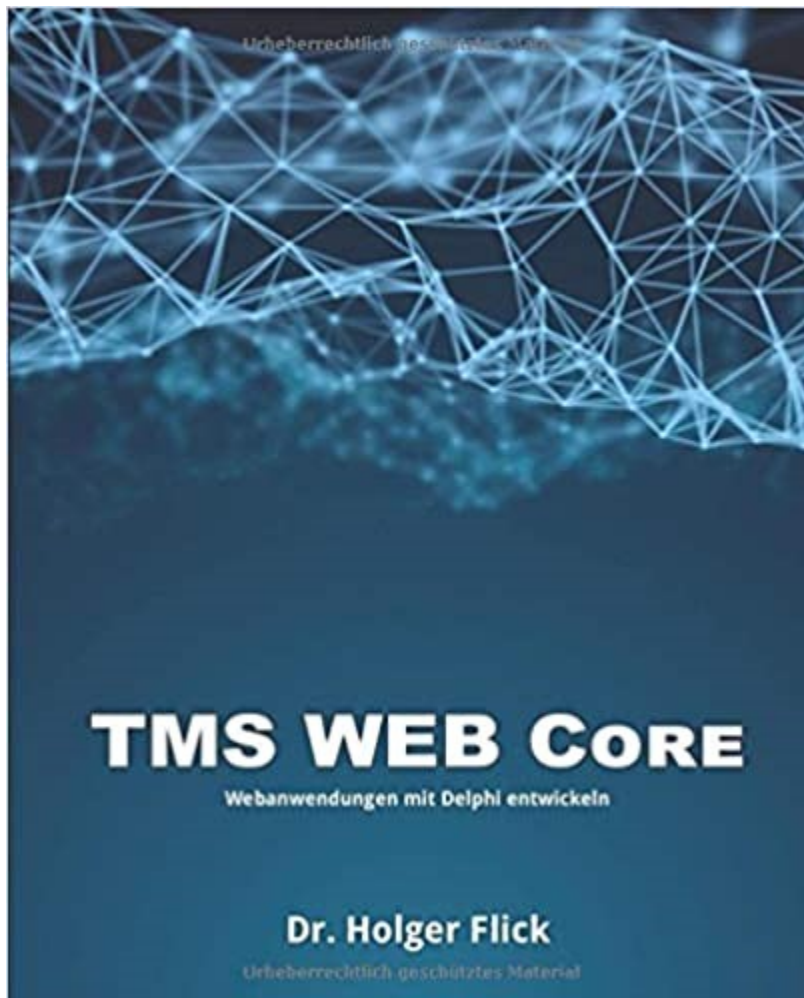
A book covering TMS WEB Core written by Dr. Holger Flick, chief evangelist at tmssoftware.com is available in both German and English language at Amazon.

The book content is summarized here:

- Detailed description of the basics, the functionality, and the transpiler (based on pas2js)
- Step-by-step creation of the first web application
- Progressive Web Applications (PWA) for offline use
- Electron applications: Cross-platform Desktop applications based on web applications
- Integration of JavaScript classes and controls
- Creating web services for databases with TMS XData
- Integration of databases with TDataset controls
- XData-specific functionality for use in web applications
- Responsive web design (form designer, HTML, CSS, Bootstrap)
- The final chapter provides a comprehensive and practical example of server and web application with Google Maps and Google Charts
- The content is suitable for both beginners and advanced developers interested in creating web applications with TMS WEB Core.



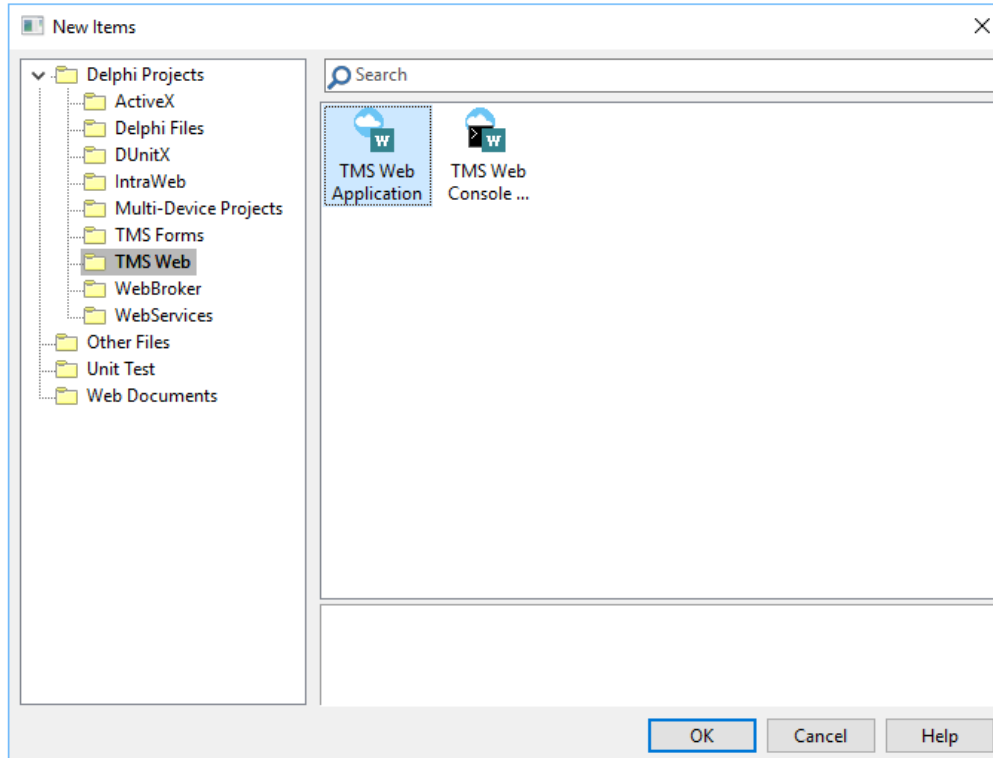
Amazon.com: <https://www.amazon.com/dp/B086G6XDGW/>



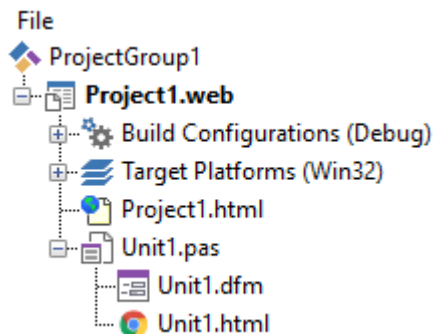
Amazon.de: <https://www.amazon.de/dp/1090700822/>

Getting started

From the Delphi IDE, choose File, New, Other and pick from the wizard either a TMS Web Application or TMS Web Console application:



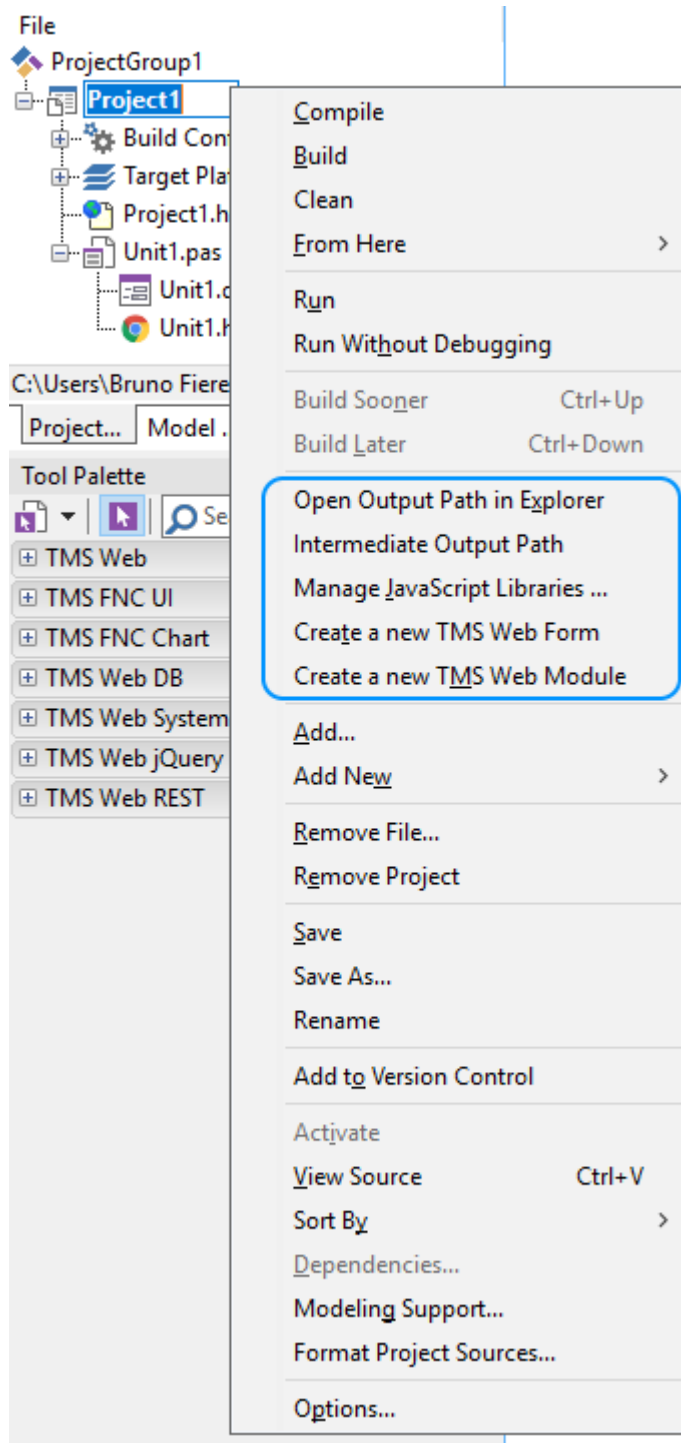
When you created a new project, it is shown in the project manager:



The project consists of a project source file and one or more form files similar to how VCL Windows applications and FMX cross platform applications work. Other than the project source file (.DPR file), there is a project HTML file. This HTML file

contains the body HTML for your application and this can include references to CSS and/or JavaScript files. For each form, there is a .PAS file, a .DFM file and a .HTML file. The .PAS file contains the user interface logic for the form and its controls. The purpose of the .DFM file is to persist the component settings and the HTML file serves as the HTML container in which the form controls will be hosted or that contains the HTML elements to which the Delphi control class instances will be mapped.

The project context menu in the IDE also shows a number of extra items:



From this context menu, the output path (where HTML, JS, CSS are generated) can be opened via Windows Explorer. It is also possible to add a new web form or data module directly from

this context menu and finally, the JavaScript Library Manager can be started. For details about the JavaScript Library Manager, see the paragraph specifically about that.

Technically, a TMS WEB Core application is at design-time in Delphi a VCL Windows application. This is for the technical reason to make use of the Delphi IDE form designer to create the web client application. The TMS WEB Core Delphi IDE plugin takes care to compile the project with all its form files with the pas2js compiler to a JavaScript (.JS) file and deploy it to a web server.

The default project source is:

```
program Project1;

uses
  WEBCLib.Forms,
  Unit1 in 'Unit1.pas' {Form1: TWebForm} (*.html);

{$R *.res}

begin
  Application.Initialize;
  Application.MainFormOnTaskbar := True;
  Application.CreateForm(TForm1, Form1);
  Application.Run;
end.
```

This looks very much the same as a standard VCL application. When the project is compiled to a JavaScript application, it can be automatically launched in the browser from the IDE. This is done via a browser launcher application (TMSWEBRunner.exe) that is configured in the IDE under Tools, Options, TMS Web, Web Runner. By default, the JavaScript application is launched in the browser that is set in the operating system as default browser. If you want to launch the application in a different browser, this can be set at project level via Project, Options, TMS Web, Browser.

To get the browser start the web client application, by default TMS WEB Core ships with a lightweight debug webserver and this is configured at install time to operate at port 8000. The web server is specified under Tools, Options, TMS Web, Web Server. See the paragraph on “Configuring TMS WEB Core settings” for more information on how to specify the web server to be used.

The default project HTML file contains:

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta $(ThemeColor)>
    <noscript>Your browser does not support JavaScript!</noscript>
    <link rel="icon" href="data:;base64,=">
    <meta $(Manifest)>
    <title>TMS Web Project</title>
    <script type="text/javascript" src="$(ProjectName).js"></script>
    <style>
    </style>
  </head>
  <body>
<meta $(BodyParameters)>
  </body>
  <script type="text/javascript">
    rtl.run();
  </script>
</html>
```

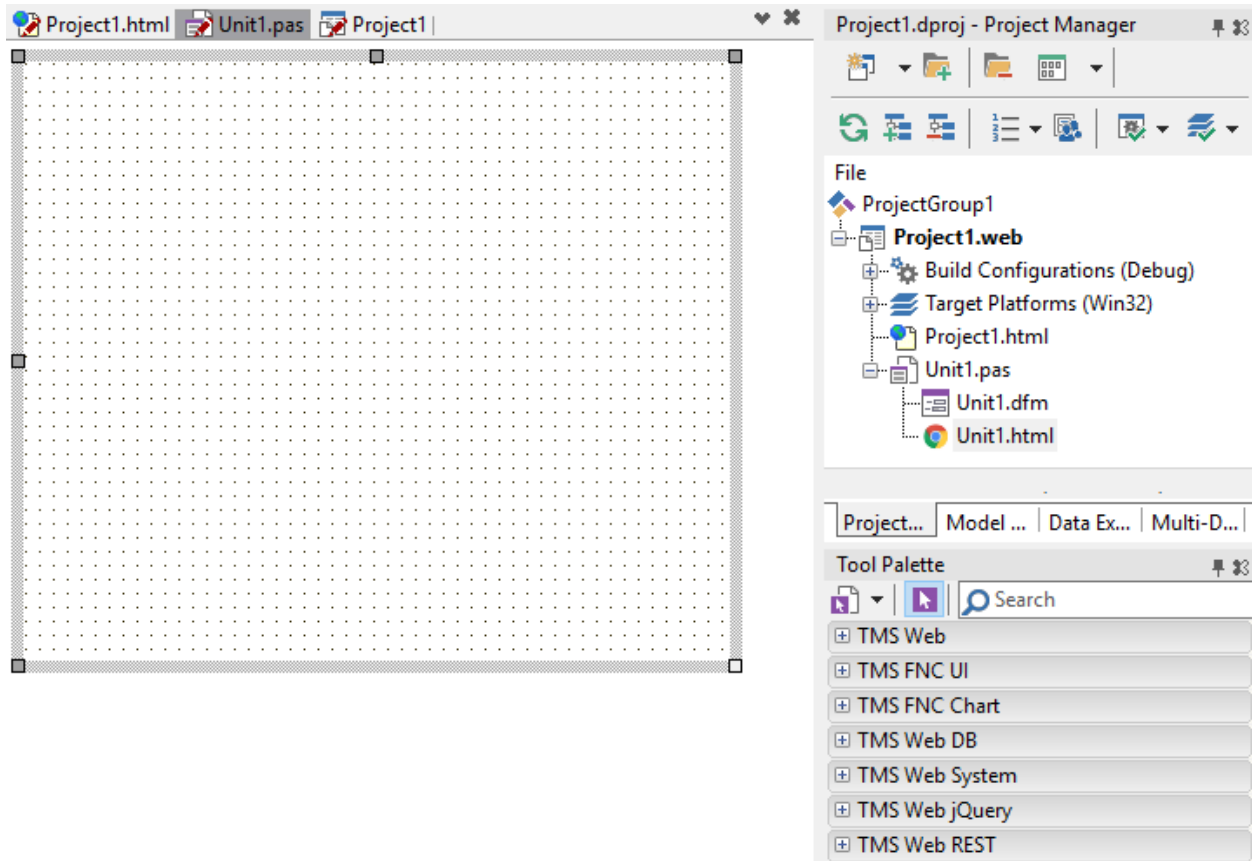
This is a HTML file specifying the HTML5 DOCTYPE. As you can see, by default, there is only one reference in the HTML file and that is to \$(ProjectName).js. The TMS WEB Core IDE plugin will in this case compile the application to Project1.js and in the deployed HTML file, this reference will be as such:

```
<script type="text/javascript" src="Project1.js"></script>
```

From the HTML file, you can see that the application is launched by

```
<script type="text/javascript">
  rtl.run();
</script>
```

When the form file in the web project is open, the IDE tool palette offers all components / controls that have been designed & registered for use with TMS WEB Core:



Just like with VCL applications, drag the controls on the form and add the UI logic code to the form file.

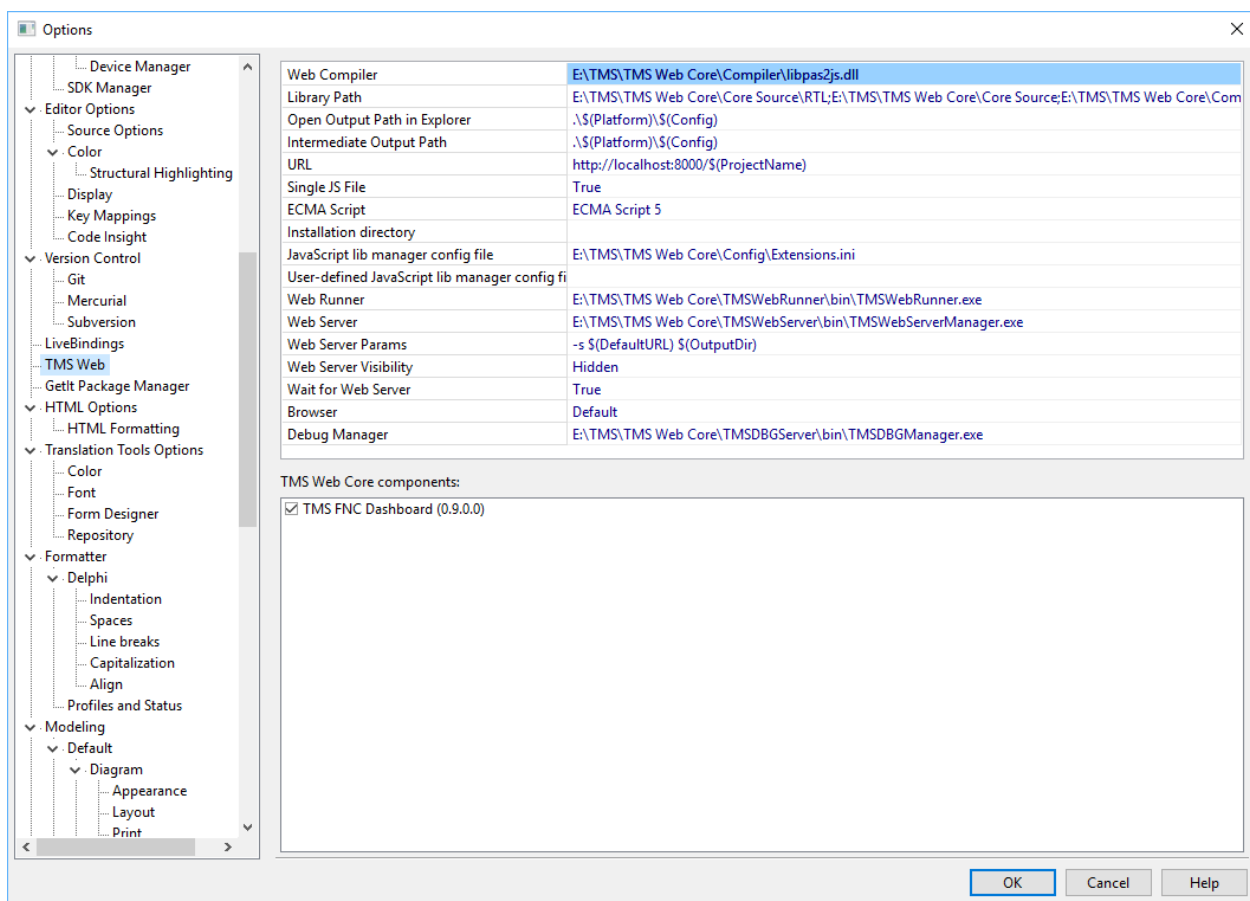
As you can see, with each form file comes a HTML file, unit1.html in this case. This is the HTML container in which the web form will be embedded. This HTML file will be loaded in to the browser document BODY when the web application launches the form. The HTML file can be directly edited from the IDE in its embedded HTML editor but it can also be edited by any web editor or by a separate web designer using his own tools.

```
<html>
  <head>
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <title>TMS Web Project</title>
    <style>
    </style>
  </head>
  <body>
  </body>
</html>
```

This is an empty HTML file. When adding UI controls to the form file, at runtime the HTML elements of which these controls are composed will be dynamically added to the HTML BODY.

Configuring TMS WEB Core settings

The configuration of TMS WEB Core can be found in the Delphi IDE under Tools, Options, TMS WEB:



The settings are:

Web Compiler: this points to the location where the Pascal to Javascript compiler (pas2js) is located.

Library path: this is the path the Pascal to Javascript compiler uses. Note that 3rd party controls can separately register paths and this does not affect the general library path.

Open Output path in Explorer: this is the default path the compiler uses to generate the resulting project Javascript (.JS) file. The default is under the project source folder \TMSWeb\Debug or \TMSWeb\Release

URL: this is the URL with which the web application can be launched via the browser. If another web server than the default TMS webserver is used, the URL can be modified here.

Single JS file: Default a single Javascript for the entire web application is generated. If this is turned off, it will be needed to specify each generated .js file reference (for each unit there is a .js file in this case) in the project HTML file.

ECMA Script: This sets the JavaScript standards level for which to generate the compiled application.

Installation directory: this holds the path where TMS WEB Core is installed. Relative to this path, the compiler searches for source files & resources.

Web Runner: this is a the path to the application that is used to start the selected browser for running the web application

Web Server: this is the path where the webserver that is used and that will be launched is located.

Use ShellExecute : this setting controls whether the IDE plugin will use Windows ShellExecute to launch the web runner application or create the process directly

Web Server Params: can contain extra command-line parameters to launch the web server

Web Server Visibility: configures whether the web server is hidden when launched or remains visible.

Wait for Web Server: when true, the IDE will wait until the webserver is effectively running before launching the browser to open the web application URL.

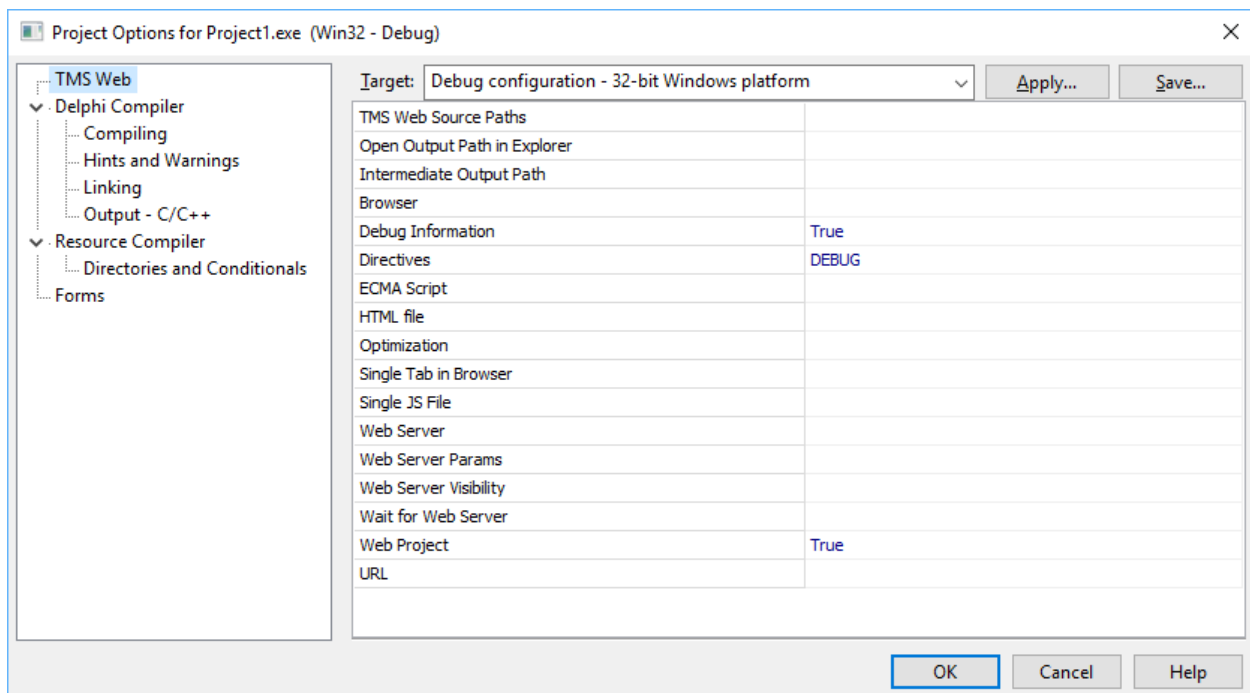
Browser: specifies if the preferred browser to launch to debug the web application.

Debug Manager: this specifies the debug tool that is used to communicate between the IDE and the browser. Via this debug manager it is possible to relaunch a web application in the same browser tab when a new version is compiled instead of launching the application in a new browser window or new browser tab.

Under the list of settings, you can find a list of installed 3rd party components for TMS WEB Core and you can check or uncheck what 3rd party component paths will be used to compile the web client application.

Configuring TMS WEB Core project settings

From the project context menu in the IDE project manager, a new pane is added to configure the options of the TMS WEB Core application:



This contains the project specific settings. By default, the TMS WEB Core general settings defined in the IDE are applied when a new project is created.

TMS Web Source Paths: optional project specific source library paths.

Open Output Path in Explorer: optional custom output path. When nothing is specified, the default output path is the folder TMSWeb\Debug or TMSWeb\Release under the project source folder

Browser: sets the browser to launch to run the application. When nothing is specified, this is the default operating system browser.

Debug Information: when true, the JavaScript debug map file is generated. This option is set by default for the Debug mode of the application.

Directives: Sets the compiler directives to use for compiling the application

ECMA Script: sets the JavaScript standards level to compile for

HTML file: sets optionally a different HTML file to launch the web application

Optimization: defines whether to compile with or without compiler optimization. When optimization is enabled (default), unused Pascal code does not get compiled to JavaScript, reducing the size of the generated JavaScript file significantly.

Single Tab in Browser: when enabled and in Debug mode, when compiling a new version of an already running web application in the browser, will result in relaunching the web application in the same browser tab as the already running application instead of opening it in a new tab.

Web Server: optionally specifies a project specific web server to use

Web Server Visibility: configures whether the web server is hidden when launched or remains visible.

Wait for Web Server: when true, the IDE will wait until the webserver is effectively running before launching the browser to open the web application URL.

Automatic versioning

If the project name is project1.dproj, the default application JavaScript file will be project1.js and is referenced in the HTML project file as

```
<script type="text/javascript" src="Project1.js"></script>
```

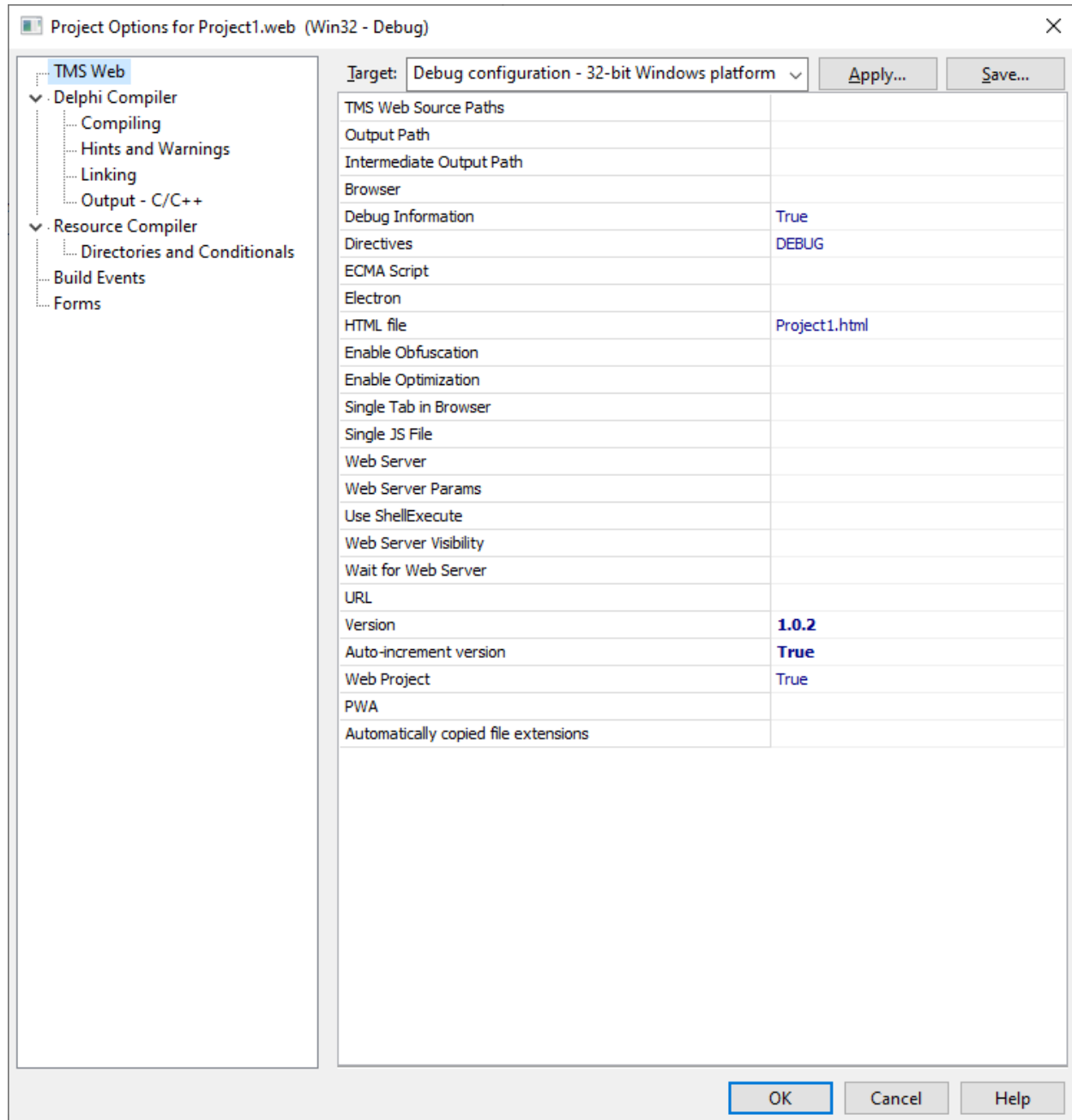
In some situations, the browser might have cached this project1.js file and the browser / web server communication fails to inform an updated project1.js should be downloaded instead of using the cached version.

To overcome this potential issue, TMS WEB Core features automatic project versioning. With this automatic versioning, each time a project is build, a new version number is generated and referenced in the HTML file, making each each project version unique and avoiding the use of a cached version when it was not expected.

Enabling automatic versioning in the project is easy. Go to project options. Set the initial version to X.Y.Z and set Auto-increment version to True

Each time a build is done, the Z-value increases.

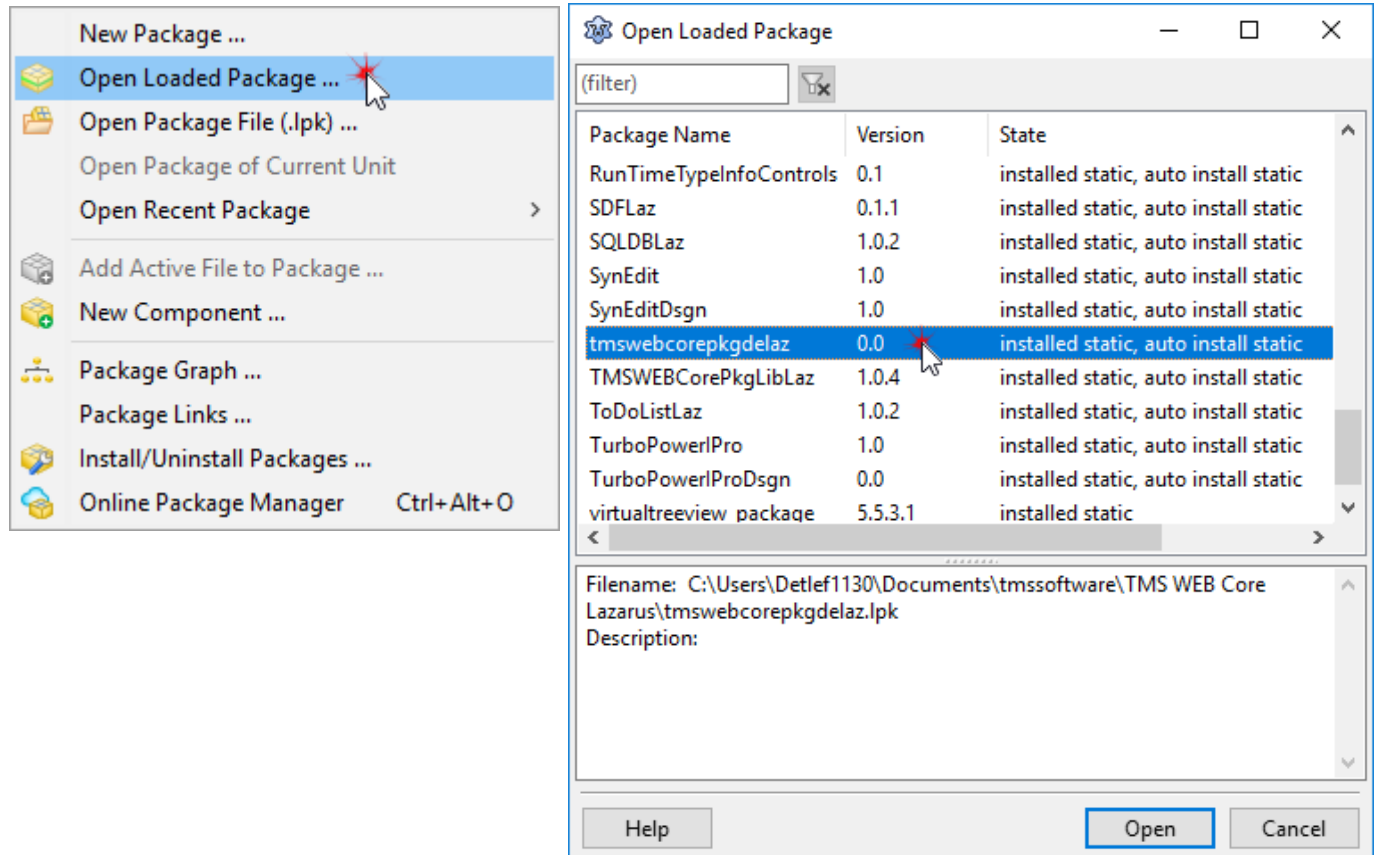
In this screenshot, you see the result after 2 builds when the initial version was set to 1.0.0:



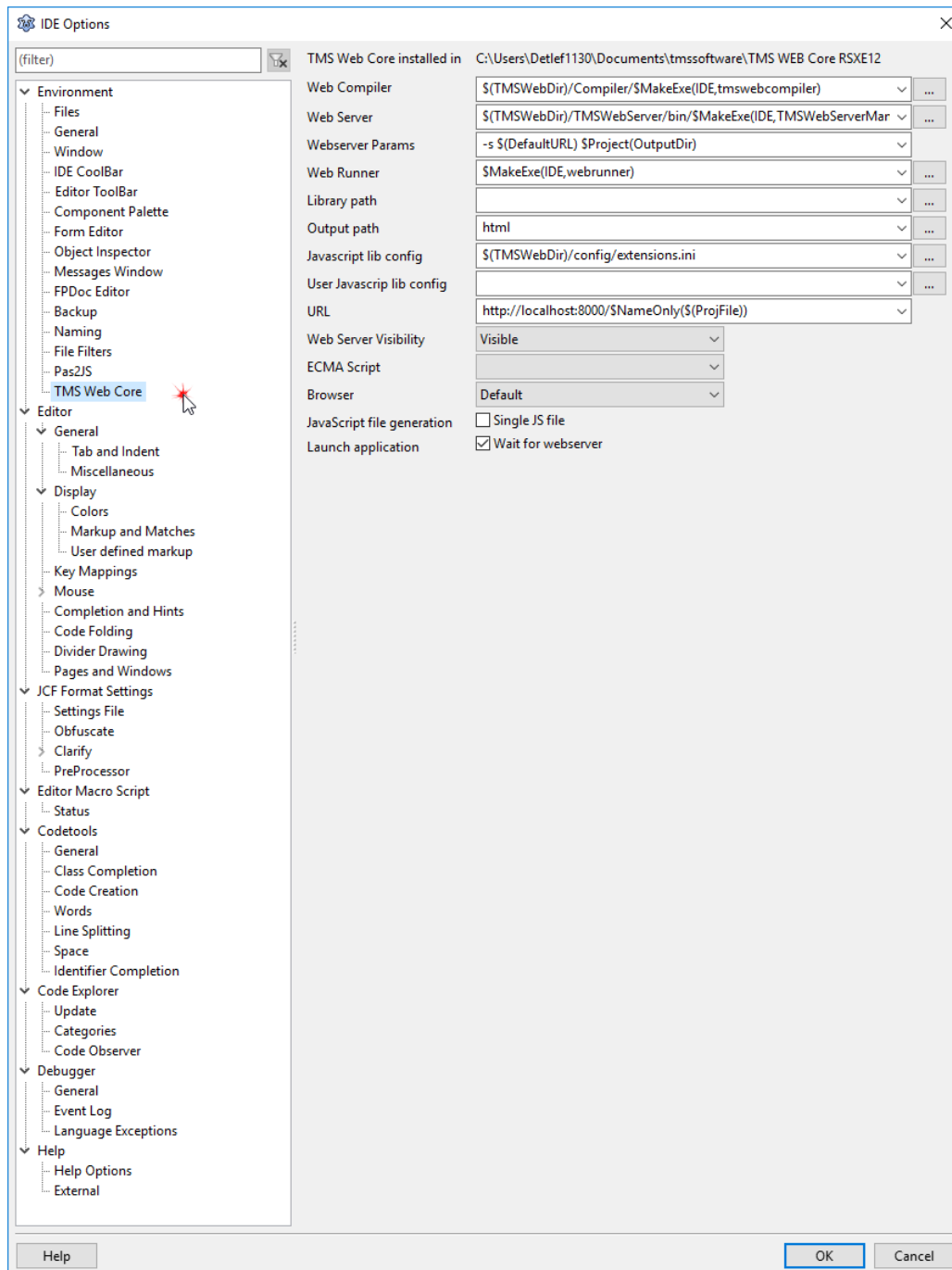
Installation for Lazarus

The compilation of the design-time package for Lazarus v2.0 needs to be done different from Lazarus v1.x

This can be configured by opening and installing the package tmswebcorepkgdelaz.lpk

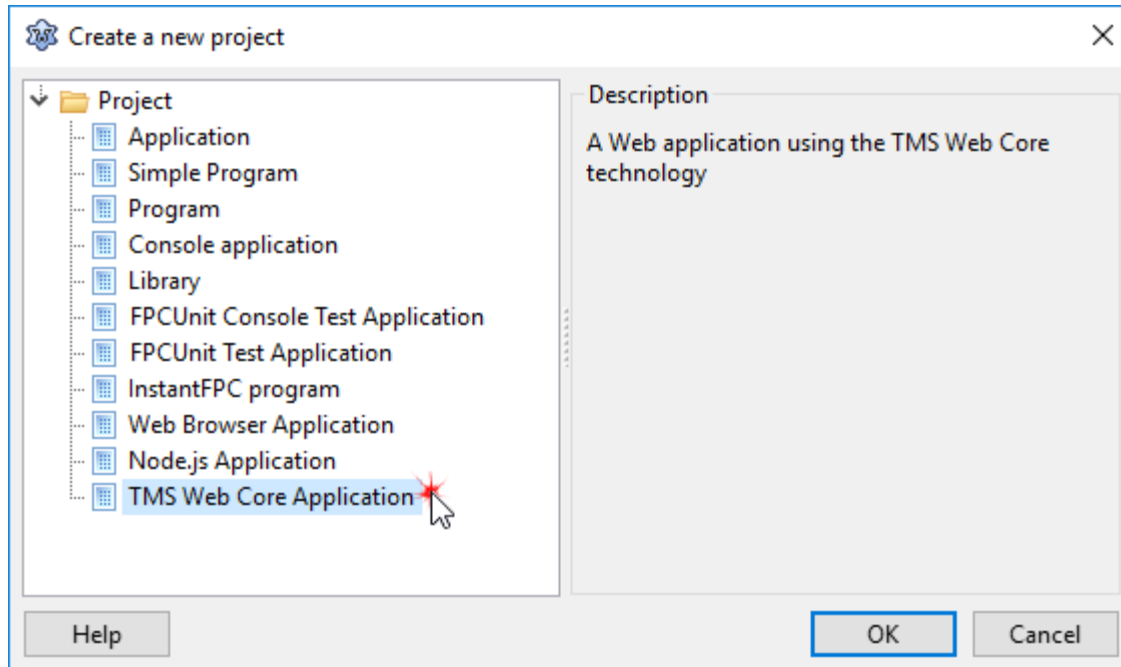


Open Tools, Environment where you can see as the last item in the list TMS WebCore where you can find all paths that need to be set:

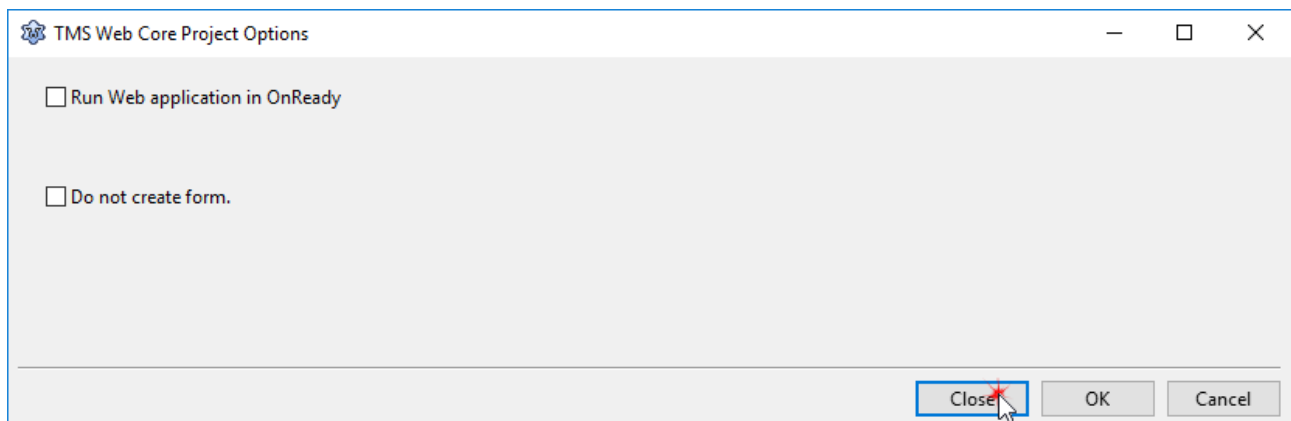


After verifying the settings, you are ready to create your first TMS WEB Core project. Go to **Project, New project**

TMS WEB Core will appear in the wizard:

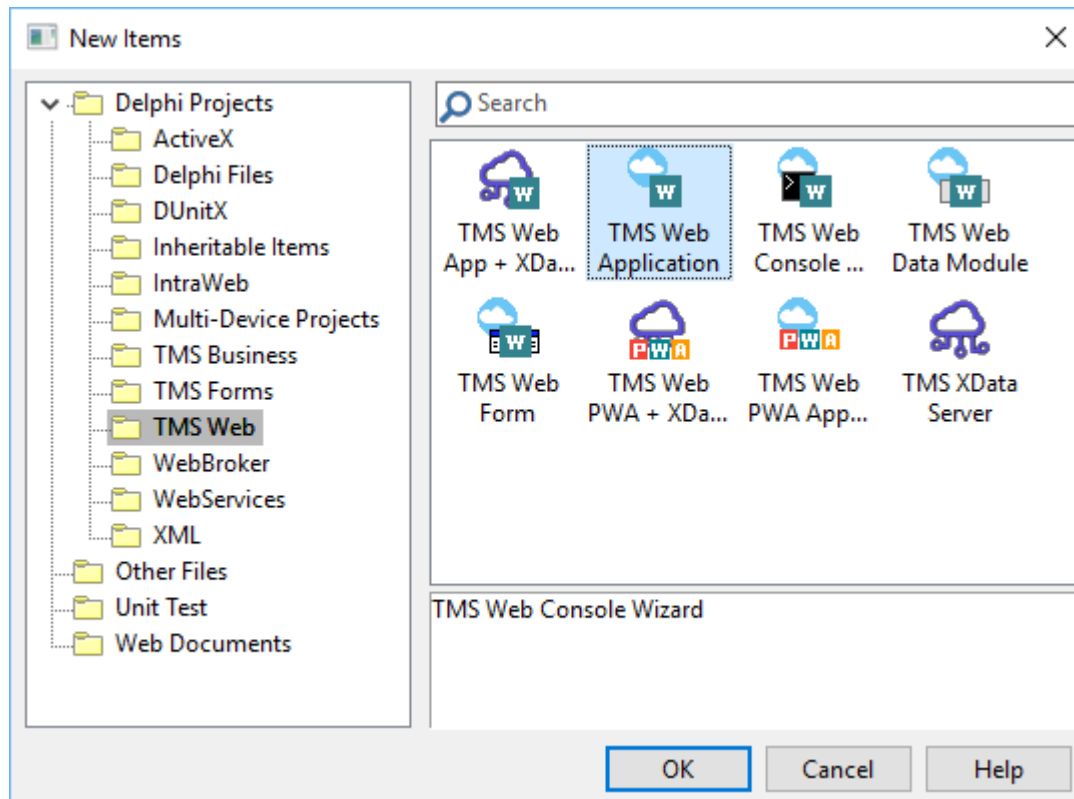


For a first basic application, leave the options unchecked and proceed to create the application.

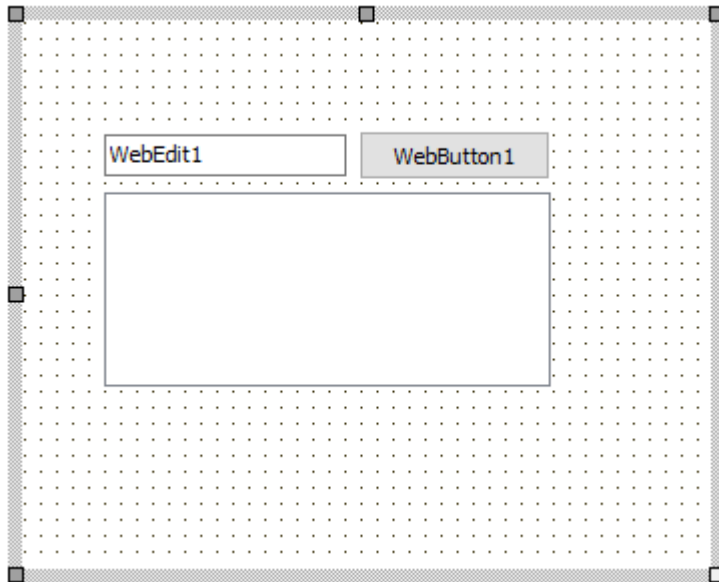


Your first TMS WEB Core application

Let's create step by step a first TMS WEB Core application. After creating a new TMS WEB Core project from the wizard



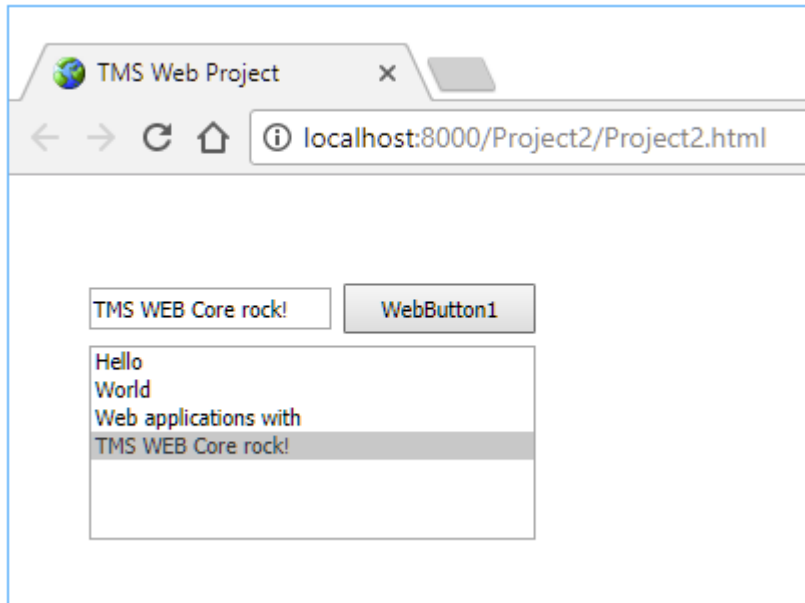
and opening the Delphi IDE form designed for the web form, let's add a TWebButton, TWebEdit and TWebListBox:



Now, let's add a WebButton event handler for OnClick:

```
procedure TForm1.WebButton1Click(Sender: TObject);  
var  
    s: string;  
begin  
    s := WebEdit1.Text;  
    WebListbox1.Items.Add(s);  
end;
```

When running this project, the result we see in the browser is:



When you have compiled the application in debug mode, the output folder contains the following files:

Name

- Project1.html
- Project1.js
- Project1.js.map
- Unit1.html

The file project1.js contains the Javascript compiled application. The file project1.html is the general project HTML file. The unit1.html is the HTML that is specific to form1 which is the default form in the project here.

An interesting file is the project1.js.map file. This is the file that facilitates debugging directly using the Delphi language from the browser. When compiling in release mode, this file is not generated.

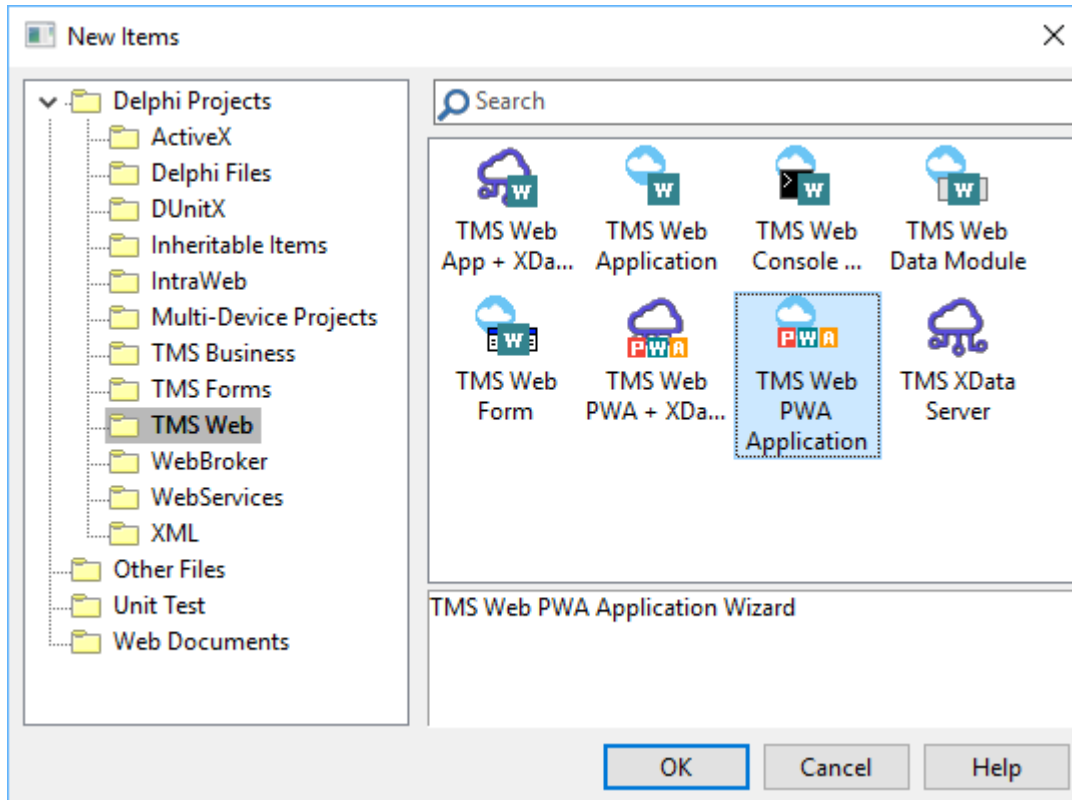
When one would want to deploy the application to a web server, all that is needed to do is put the files project1.html / project1.js and unit1.html in a folder on a web server.

Your first TMS WEB Core progressive web application

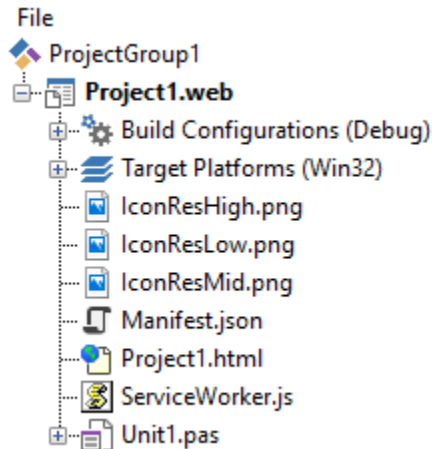
It is also possible to automatically create a progressive web application (PWA) from the IDE wizard. A progressive web application is a web application designed to adapt itself to online/offline situations, to various device types and most importantly, to let itself install similar to a native application on the desktop and start from a desktop icon. More information about progressive web applications can be found here:

<https://developers.google.com/web/progressive-web-apps/>

To create a new progressive web application from TMS WEB Core, select the icon “TMS Web PWA Application” from the wizard:



At first sight, it looks like this generates the template for a regular TMS WEB Core web application. However, several important additional files are generated: the manifest file, the JavaScript serviceworker file and application icons in different sizes:



The manifest file is the file that contains the name, description, icons and general information of your progressive web application. This manifest must conform to the standard:

<https://developers.google.com/web/fundamentals/web-app-manifest/> When it is available, it allows the browser to identify the application as progressive web application and offer to install the application from the desktop.

In a TMS WEB Core, the manifest is automatically linked from the project main HTML file via:

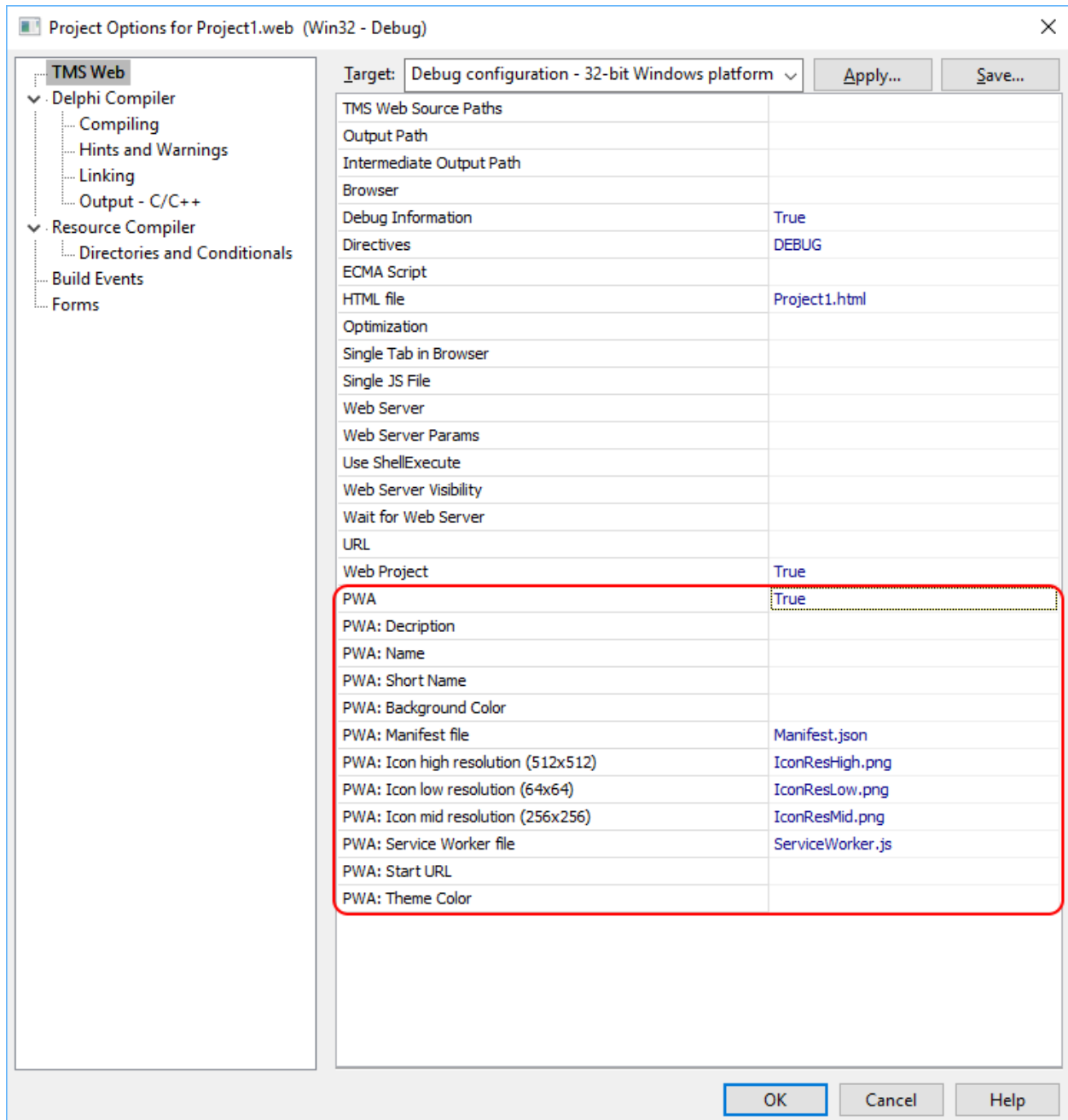
```
<link rel=manifest href="Manifest.json"/>
```

The serviceworker is registered and invoked as well from the main HTML script

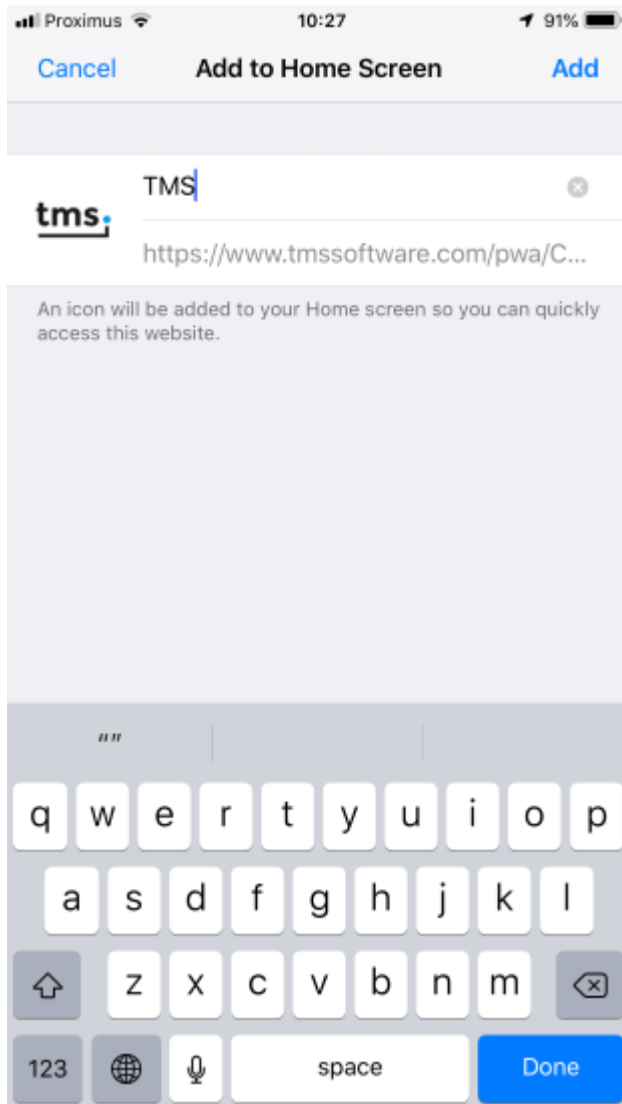
```
<script type="text/javascript">
  if ("serviceWorker" in navigator)
  {
    navigator.serviceWorker.register("ServiceWorker.js").then(
      function(ARegistration)
      {}).catch(
        function(AErr)
        {
          console.log("TMS WEB Core service worker registration failed", AErr);
        });
  }
</script>
```

While the default generated serviceworker.js contains all functionality to automatically cache your entire TMS WEB Core application for offline use, it can be further customized by editing this file in JavaScript.

The manifest file content can be customized directly from the project options:



When the progressive web application is launched from the browser via its URL, browsers supporting progressive web applications (Safare on iOS, Chrome on Android), will show a dialog upon launching to offer the possibility to add the application icons to the home screen:



When a progressive web application is used, the Application singleton object returns the online/offline state of the application and will also trigger an event when the online/offline state changes.

Check the property

```
Application.IsOnline: boolean
```

to check whether the application is online or offline.

Or attach an event handler to:

```
Application.OnOnlineChange: TApplicationOnlineChangeEvent
```

with

```
TOnlineStatus = (osOnline, osOffline);
```

```
TApplicationOnlineChangeEvent = procedure(Sender: TObject; AStatus:  
TOnlineStatus) of object;
```

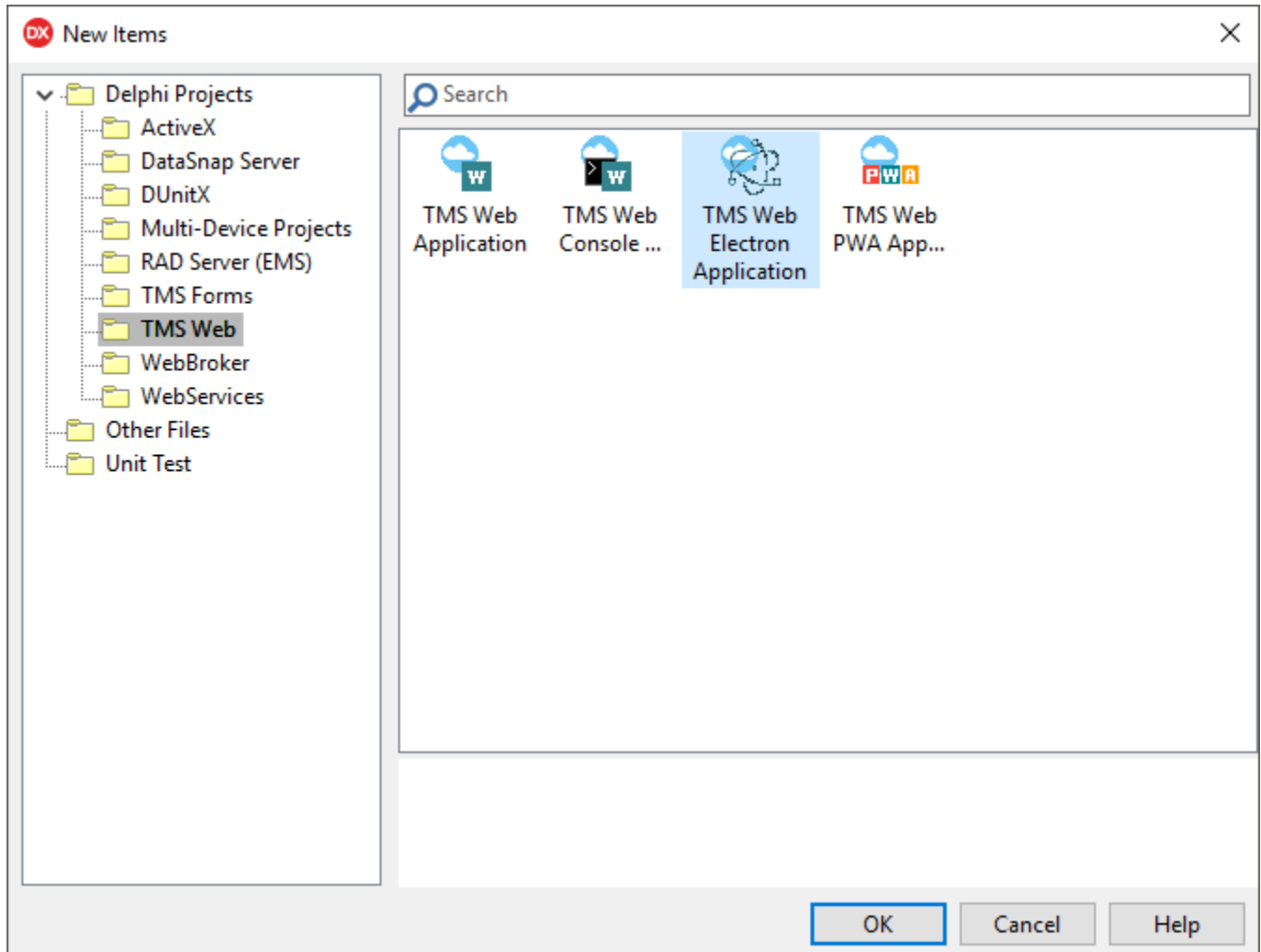
This event handler will be triggered when the internet connection availability changes on the device where the application is run.

Your first TMS WEB Core Electron Application

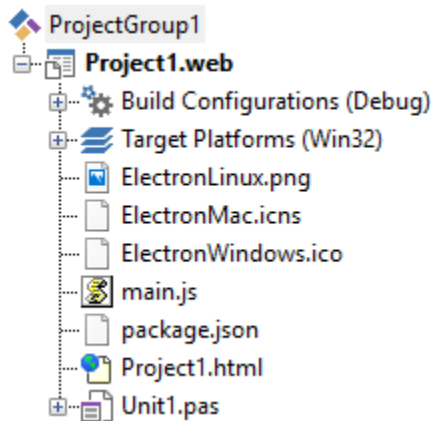
TMS WEB Core allow to create cross platform Electron applications. These are packaged web applications that can be deployed as executable code for Windows, macOS and Linux. You can learn about the exciting Electron framework at: <https://electronjs.org/>

It is the Electron framework that offers a large API to take advantage of operating system features such as application menu, notifications, local file access, ... Your TMS WEB Core application gets compiled to JavaScript and it is the Electron packager that turns the compiled result into an executable for the 3 operating systems: Windows, macOS, Linux. TMS WEB Core integrates all these steps for you from the IDE for Windows & Linux. For macOS, the packager needs to be run separately from a macOS operating system.

To create a new Electron application from TMS WEB Core, select the “TMS Web Electron Application” from the wizard:



It generates a project that is similar to a TMS Web PWA Application, but instead of the manifest and serviceworker files, it has generated a main javascript file, a package file and 3 icons for the different platforms:



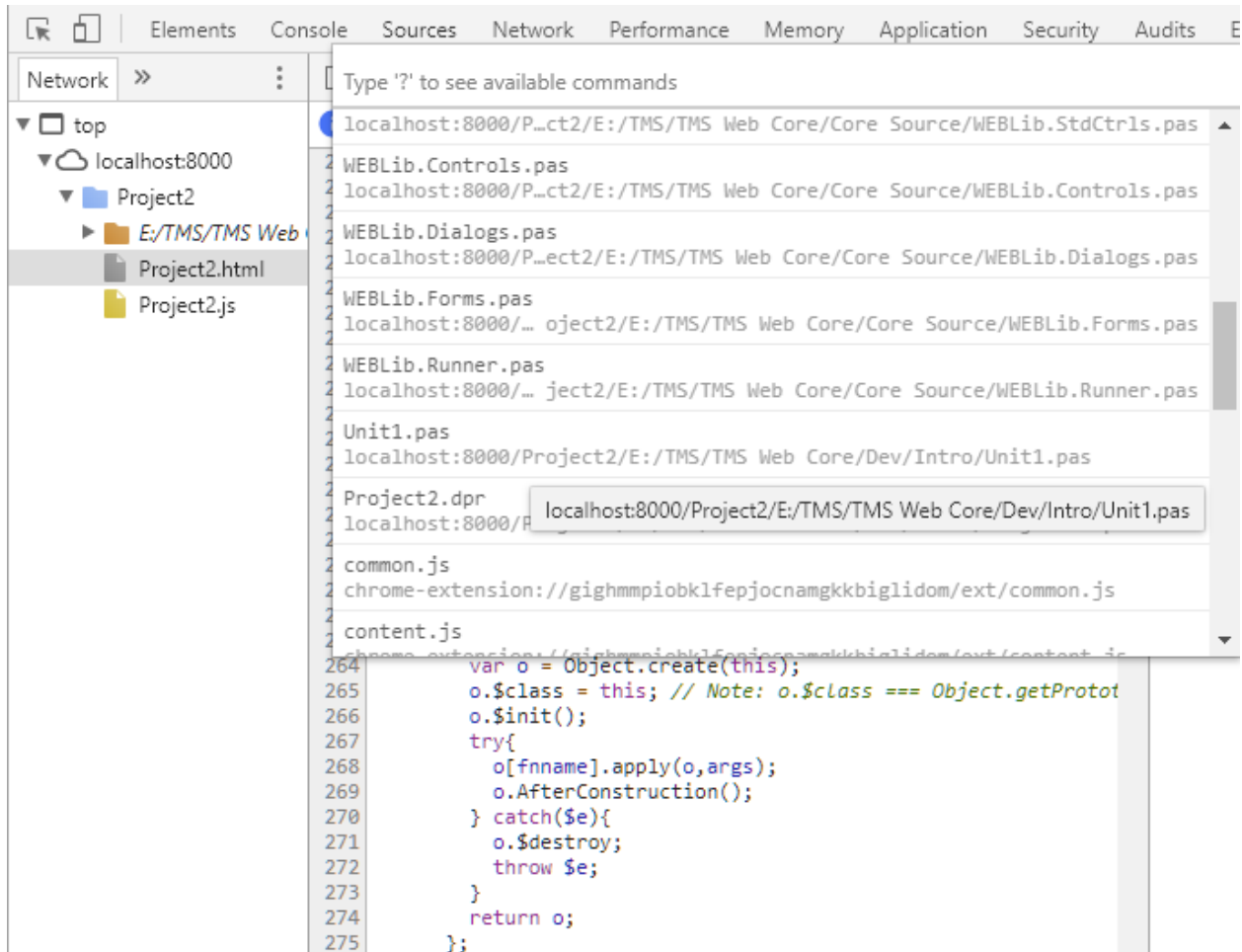
For Electron application development with TMS WEB Core, a whole range of components is available that let you take advantage of the Electron APIs for interfacing with the operating system. This is covered in the chapter specifically about Electron.

Debugging

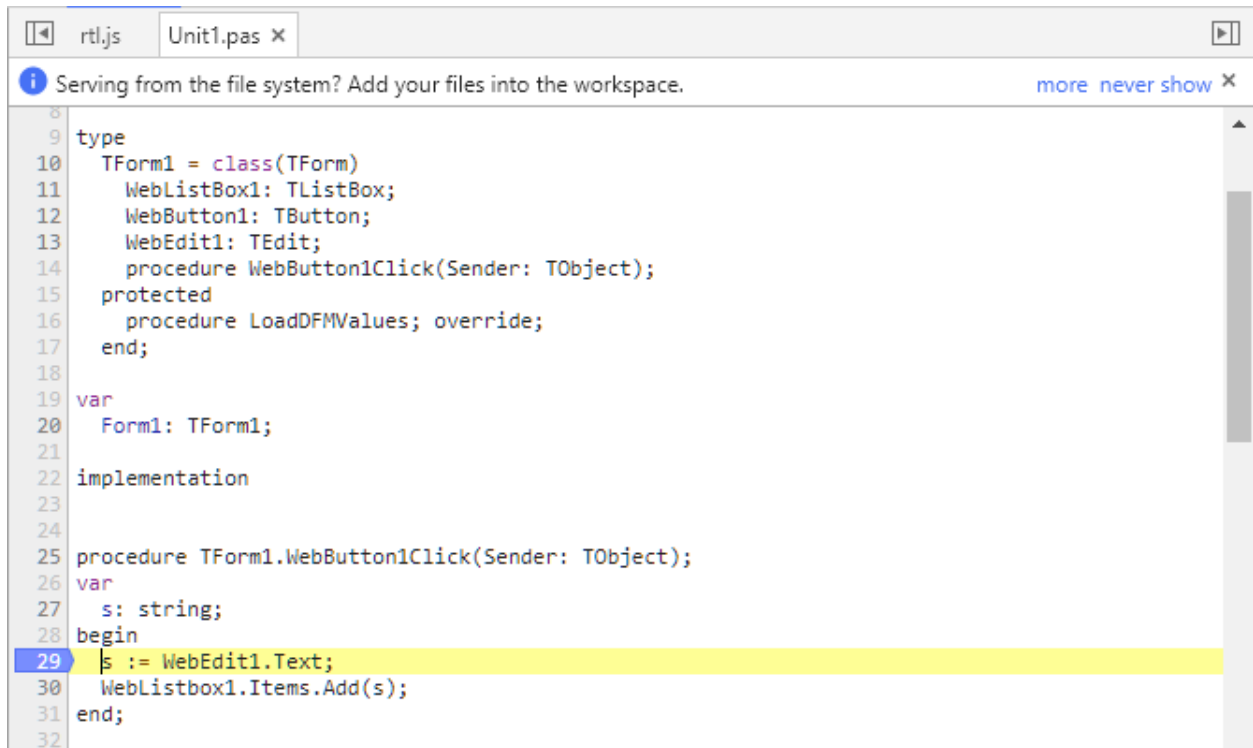
As explained in the previous paragraph, when compiling in the Delphi IDE in debug mode, the extra file `project1.js.map` is generated to offer the capability to debug the application directly from Delphi code in the browser. This capability is supported in both the Chrome and Firefox browser.

To start the debugger, press F12 from the browser and go to the Sources tab. On the source window, press shortcut Ctrl-P and you get to see the file list of all files involved in the project.

From the file list, pick `unit1.pas`:



After picking this file `unit1.pas`, you can add breakpoints by clicking in the line number gutter:

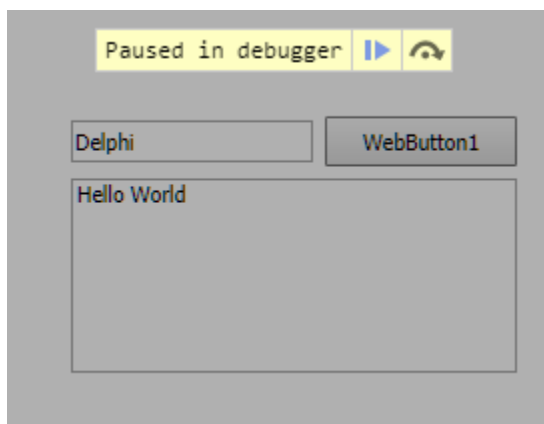


```

8
9 type
10   TForm1 = class(TForm)
11     WebListbox1: TListBox;
12     WebButton1: TButton;
13     WebEdit1: TEdit;
14     procedure WebButton1Click(Sender: TObject);
15   protected
16     procedure LoadDFMValues; override;
17   end;
18
19 var
20   Form1: TForm1;
21
22 implementation
23
24
25 procedure TForm1.WebButton1Click(Sender: TObject);
26 var
27   s: string;
28 begin
29   s := WebEdit1.Text;
30   WebListbox1.Items.Add(s);
31 end;
32

```

Now, adding a value in the TWebEdit control and pressing the TWebButton triggers the breakpoint:



and as you can see in the browser debugger, you can step Delphi line by Delphi line through the code and see the values of Delphi variables directly in the browser debugger:

```

24
25 procedure TForm1.WebButton1Click(Sender: TObject); Sender = {$class: {...}, FOwner: {...}, FName: "WebBut
26 var
27   "Delphi" s; s = "Delphi"
28 begin
29   s := WebEdit1.Text; s = "Delphi"
30   WebListBox1.Items.Add(s);
31 end;
32

```

This shows how easy and comfortable it is to debug TMS WEB Core application using the Delphi language directly from the browser.

To make it even more convenient to debug your TMS WEB Core applications in the browser, when you compile your application in Debug mode and you have added a breakpoint in the Delphi IDE, this is automatically converted to a debugger breakpoint in the browser.

For example, set the breakpoint in the Delphi IDE:

```

- procedure TForm1.WebButton1Click(Sender: TObject);
- var
-   s: string;
- begin
-   s := WebEdit1.Text;
-   WebListBox1.Items.Add(s);
- end;

```

and run the web application in the browser results in:

```

25 procedure TForm1.WebButton1Click(Sender: TObject); Sender = {FOwnerInterface: {...}, FOwner: {...},
26 var
27   s: string; s = ""
28 begin
29   asm debugger; end;
30   s := WebEdit1.Text;
31   WebListBox1.Items.Add(s);
32 end;
33

```

So, the browser debugger is automatically forced to stop just before the line where the breakpoint was set in the IDE. This saves you from locating the code in the browser console and set any breakpoints again from there.

An alternative to setting breakpoints via the IDE, is to set a breakpoint by inserting the identifier {BP} in the code. For every line in the Pascal code that contains {BP}, a JavaScript code line "debugger" will be inserted.

Pascal to JavaScript Compiler

For creating the single-page JavaScript application, the Pascal code of your project is compiled to JavaScript and this JavaScript application runs in the browser. Typically, a connection to the server will be made by HTTP REST requests or via WebSocket communication. To compile the Pascal code to JavaScript code, the pas2js compiler is used that is an open-source project and builds on years of experience of the FPC compiler team. More information about the pas2js project can be found here: <http://wiki.freepascal.org/pas2js>

At this moment, this support for the Pascal language is highly compatible with the Delphi language. Pas2js v2.0 introduces advanced features such as attributes, generics, type helpers, support for JavaScript await, promises and much more. Some of the newest Delphi language features are not yet supported in pas2js v2.0 but on the radar for future releases:

- Advanced Records
- Advanced RTTI
- Inplace variables

For more details about the capabilities of the pas2js compiler, please refer to <http://wiki.freepascal.org/pas2js#Compiler>

TMS WEB Core ships with a validated version of the pas2js compiler. We recommend using the pas2js compiler included in the TMS WEB Core distribution as this is the version we test & approve our entire framework and IDE integration with.

RTL

Equally important to move existing VCL or FMX code bases to the web is the support for RTL. A huge part of the Delphi RTL is available and delivered with the compiler. This includes now:

There is a basic Object Pascal RTL, several units from the FPC Packages are also available

- system
- sysutils
- math
- strutils
- rtlconst
- classes
- contnrs
- typinfo
- objpas
- dateutils
- DB
- js (javascript system objects)
- web (browser provided objects)
- libjquery

For more information about RTL support, please see: <http://wiki.freepascal.org/pas2js#RTL>

Preprocessor

As a webbrowser does not know the concept of DFM files to load form configuration from and as the form designer in the Delphi IDE is technically a VCL form, there is a preprocessing step before compiling the code to JavaScript. This preprocessing step handles the conversion from DFM file to code for initialization of the form. The pre-processor will also convert the unit namespaces from VCL to WEBLib. This unit namespace conversion not only applies to unit names but also when unit name prefixes are explicitly used for types. If there is a special reason to disallow the pre-processor to skip lines in the code, prefix these lines with the {NOPP} specifier.

So, the line

VCL.StdCtrls.TLabel

will be converted to

WEBLib.StdCtrls.TLabel

The line

{NOPP} VCL.MyUnit.MyType

will remain as-is before it is being compiled, i.e.

VCL.MyUnit.MyType

Command-line compiler

It is possible to build TMS WEB Core web client projects outside the IDE. This is done via the command-line compiler. The command-line compiler is located in the CommandLineCompiler subfolder of the install folder. It contains a Windows version and a Linux version. The name is: TMSWebCompiler.

Running the command-line compiler for a project is done with:

```
tmswebcompiler.exe /ParseDprojFile /ProjectFile:myproject.dproj
```

The compiler has various flags that can be used. Frequently used flags will be to choose the config from the project, i.e.

```
tmswebcompiler.exe /ParseDprojFile /ProjectFile:myproject.dproj /Config=Release
```

selects to compile in release mode.

To see a full list of command-line parameters, use

```
tmswebcompiler.exe -help
```

Note that several of these command-line parameters will override the settings used in the .DPROJ file.

This is an extensive list of the parameters:

```
CompilerBin, Pas2JS dll, example /CompilerBin:c:\temp\libpas2js.dll
Config, configuration, example /Config:Release
CopyFiles, files that are copied to the html dir, example
/CopyFiles:"c:\temp\picture.bmp;c:\temp\styles.css"
Debug, example /Debug
Compiler defines, example /Defines:RELEASE;DEBUG
Compile all dproj files which can be found here, example
/DprSearchPath:c:\temp
EcmaScript, 0 = default, 1 = EcmaScript5, 2 = EcmaScript6, example
/EcmaScript:1
Version of the generated Electron binary, example /ElectronAppVersion:"1.0.0"
Target-System, Win32: 2, Win64: 3, Linux32:4, Linux64:5, MacOS32: 6, MacOS64:
7, example /ElectronBuild:3
Name of the Electron Linux icon file, example
/ElectronIconLinuxFileName:"IconLinux.png"
```

Name of the Electron Mac icon file, example
/ElectronIconMacFileName:"IconMac.icns"
Name of the Electron Windows icon file, example
/ElectronIconWindowsFileName:"IconWindows.ico"
Name of the Electron main.js file, example /ElectronMainJSName:"main.js"
Name of the Electron package.json file, example
/ElectronPackageJSONName:"package.json"
Electron Application, example /Electron
Version of Electron with which the binary should be created, example
/ElectronVersion:"6.0.0"
help or no switch, example /help
HiddenMessages, example /HiddenMessages:123,456
HTMLOutputDir, html and JavaScript output directory, example
/HTMLOutputDir:c:\temp
IncSearchPaths, include search paths, example
/IncSearchPaths:c:\temp;c:\temp2
Set the language of the compiler (0 = English, 1 = German, 2 = French),
example /Language:0
Absolute paths in the map file, example /MapFileAbsolutePath
No XSSIProtection, example /NoXSSIProtection
Obfuscation, /Obfuscation
Optimization, optimization or not, 0 = no optimization, 1 = optimization,
example /Optimization:1
Parse the dproj file, example /ParseDprojFile
ProjectFile, name of the project file, example
/ProjectFile:c:\temp\project.dpr
ProjectHTMLFile, name of the project html file, example
/ProjectHTMLFile:c:\temp\project.html
PWA BackgroundColor, example /PWABackgroundColor:Black
PWA Description, example /PWADescription:Description
PWA Icon Res High, example /PWAIconResHigh:Icon.png
PWA Icon Res Low, example /PWAIconResLow:Icon.png
PWA Icon Res Mid, example /PWAIconResMid:Icon.png
PWA Manifestfile, example /PWAManifest:Manifest.json
PWA Name, example /PWAName:Name
PWA ServiceWorkerfile, example /PWAServiceWorker:ServiceWorker.js
PWA ShortName, example /PWAShortName:ShortName
PWA StartURL, example /PWAStartURL:127.0.0.1/Project1.html
Progressive Web Application, example /PWA
PWA ThemeColor, example /PWAThemeColor:Black
ShowConditionals, example /ShowConditionals
ShowDebugNotes, example /ShowDebugNotes
ShowErrors, example /ShowErrors
ShowEverything, example /ShowEverything
ShowHints, example /ShowHints
ShowInfo, example /ShowInfo
ShowLineNumbers, example /ShowLineNumbers

ShowMessageNumbers, example /ShowMessageNumbers
ShowNotes, example /ShowNotes
ShowNothing, example /ShowNothing
ShowTriedFiles, example /ShowTriedFiles
ShowUsedTools, example /ShowUsedTools
ShowWarnings, example /ShowWarnings
SingleInstance, single tab in the browser, should only be used for debug
purpose, example /SingleInstance
SingleJS, single JavaScript file or not, example /SingleJS
UnitSearchPaths, unit search paths, example /UnitSearchPaths:c:\temp;c:\temp2
Verbose, example /Verbose
Version, example /Version:1.0.0

Utility functions

The unit `WEBLib.WebTools` contains several helper functions that can be handy.

The list of available utility functions is:

```
procedure MessageBeep(AType: integer);
```

Method with a VCL compatible signature playing a beep in the browser.

```
procedure OutputDebugString(const s: string);
```

Sends the string to the browser event log

```
function GetTickCount: longint;
```

Returns the number of ticks since browser start in milliseconds

```
function GetQueryParam(AName: string): string;
```

Returns the URL query parameter value for the URL with which the web application was started.
Example:

<https://www.myserver.com/mysite?user=Admin>

with return 'Admin' for `GetQueryParam('user')`;

```
function HasQueryParam(AName: string; var AValue: string): boolean;
```

Returns true if a specific query parameter is present in the URL with which the application was launched.

```
function GetLocaleShortDateFormat(ALocale: string = ''): string;
```

Gets the short date format according to the browser locale

```
function GetLocaleLongDayName(DayOfWeek: integer; ALocale: string = ''): string;
```

Gets the long day name for a specific day in the week according to the browser locale

```
function GetLocaleShortDayName(DayOfWeek: integer; ALocale: string = ''): string;
```

Gets the short day name for a specific day in the week according to the browser locale

```
function GetLocaleLongMonthName(Month: integer; ALocale: string = ''): string;
```

Gets the long month name for a specific day in the week according to the browser locale

```
function GetLocaleShortMonthName(Month: integer; ALocale: string = ''): string;
```

Gets the short month name for a specific month in the year according to the browser locale

```
function GetLocaleDecimalSeparator(ALocale: string = ''): string;
```

Gets the decimal separator character according to the browser locale or the specified locale

```
function GetLocaleThousandSeparator(ALocale: string = ''): string;
```

Gets the thousand separator character according to the browser locale or the specified locale

```
function GetLocaleCurrency(ALocale: string = ''): string;
```

Gets the currency according to the browser locale or the specified locale

```
function LocaleFormatCurrency(Value: double; ACurrency: string; ALocale: string = ''): string;
```

Formats a value with a currency according to the browser locale or the specified locale

```
function GetBrowserLocale: string;
```

Retrieves the browser locale as string. See appendix for possible locale names.

```
function ProcessAccelerator(AValue: string; var Accelerator: string): string;
```

Converts a string using accelerator keys (i.e. characters preceded by &) as underlined HTML text. The Accelerator var parameter is set to the accelerator key value.

Example:

'My &Button'

will be converted to

'My utton' and the Accelerator var parameter will be set to 'B'

```
function GetBase64Image(AImage: TJSHTMLElement): string
```

Retrieves the image data as base64 encoded string

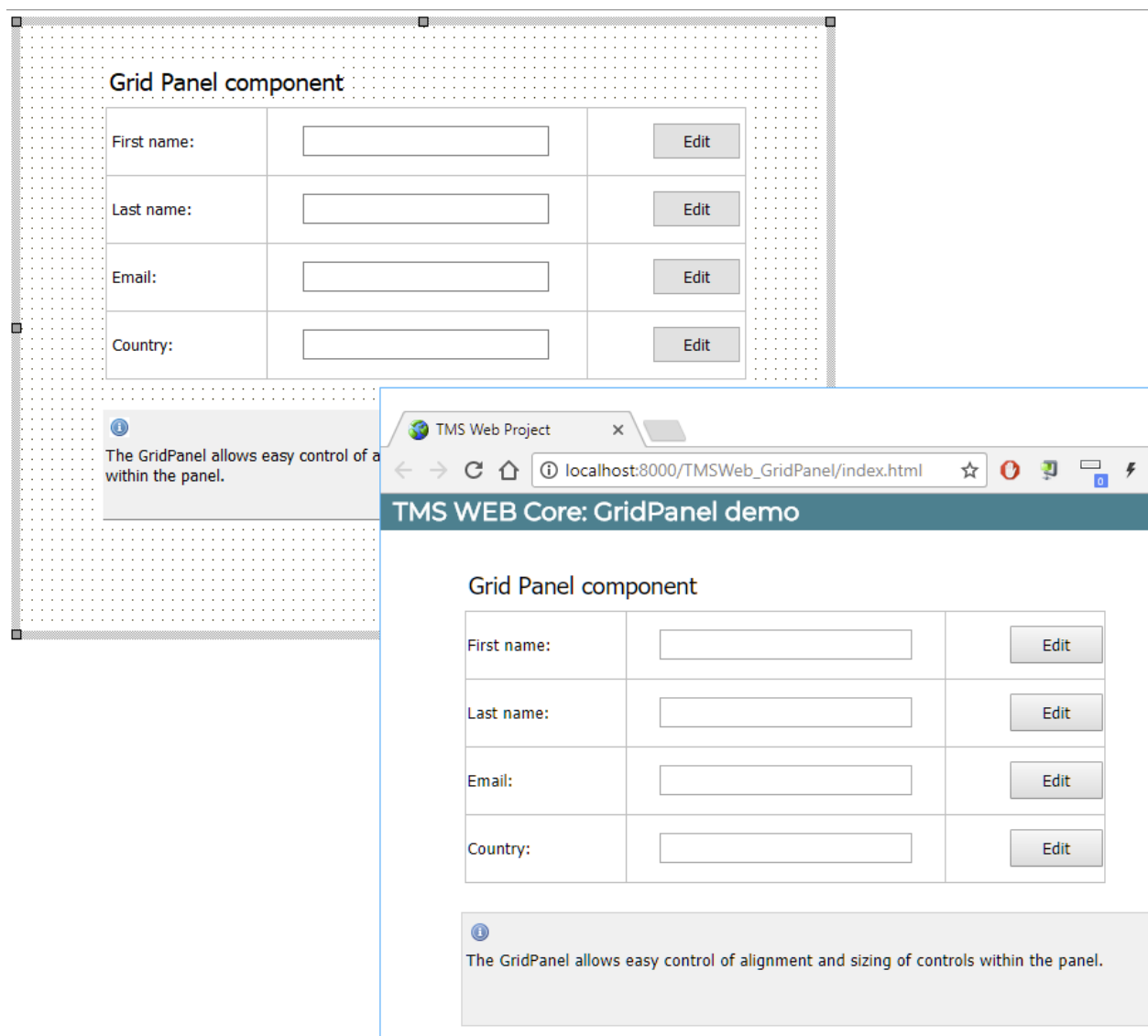
```
procedure DebugBreak;
```

Sets a breakpoint in the code at the line where DebugBreak is called.

Page Design

Absolute positioning

By default, the Delphi form designer serves as a WYSIWYG design surface for your web application forms. This means that the UI controls on the Delphi form will appear absolute positioned on the web page. For page layout & organization, there are the typical Delphi container controls like a panel, groupbox, scrollbox, gridpanel.



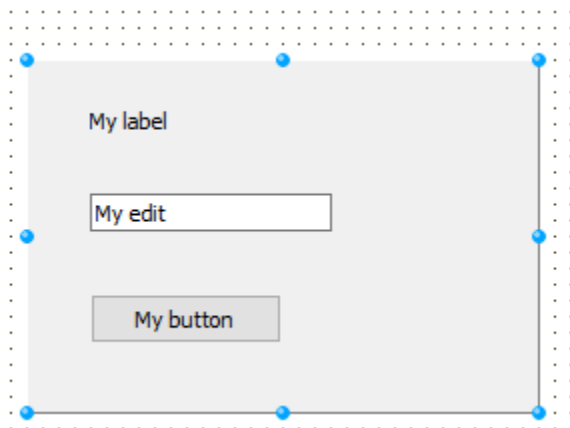
The parent/child relationship of the Delphi controls is also reflected on the produced web pages. Additional facilities like control alignment, anchoring, grid panel and a splitter control are available to let you and the end user control the layout of the pages. In this default mode, everything is as such very familiar to Delphi developers and users of Delphi VCL Windows applications and sometimes this similarity is desirable.

Relative positioning

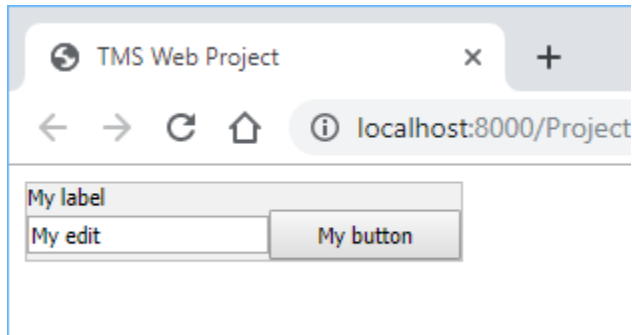
Controls can also be added to the designer and positioning set to relative position. This is set by the property `Control.ElementPosition` to `epRelative`. In this mode, coordinates for the control are not rendered. It is up to the browser DOM to determine the layout. If for the Control the `HeightStyle` and `WidthStyle` are set to `ssAuto`, also the DOM will determine the runtime size of the control. There is one very important consideration with relative positioned controls and that is control order. The relative ordering of controls is set by the `Control.ChildOrder` property. When `Control.ElementPosition` is `epRelative`, the `Control.ChildOrder` is used to control the ordering of the HTML elements of the control in the parent. The control with `Control.ChildOrder` set to 0 will be the first control under the parent HTML element hierarchy, the next control will be the control with `Control.ChildOrder` set to 1 and so on ...

Example:

In the designer, there is a panel with `ElementPosition = epRelative` and `WidthStyle`, `HeightStyle` are set to `ssAuto`. It contains 3 child controls, a label, edit and button with `ElementPosition` set to `epRelative` as well. The label's `ChildOrder` is set to 0, the edit's `ChildOrder` to 1 and the button's `ChildOrder` to 2.



The result in the browser is:



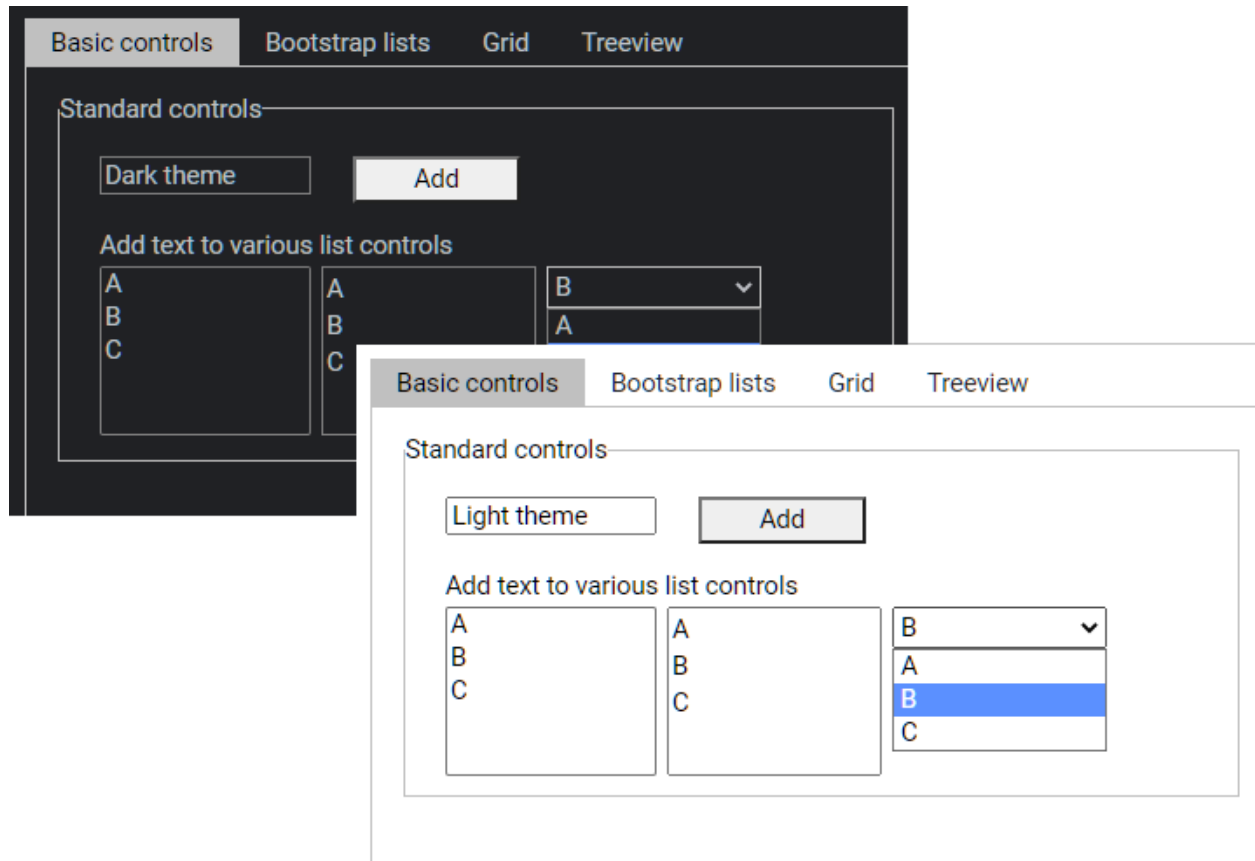
The corresponding HTML will be: a HTML SPAN element for the panel, a DIV element with child HTML LABEL element for the label. A HTML INPUT element for the edit control and a HTML BUTTON element for the button:

```
<SPAN>
<DIV><LABEL></LABEL></DIV>
<INPUT type="TEXT">
<BUTTON type="BUTTON">
</SPAN>
```

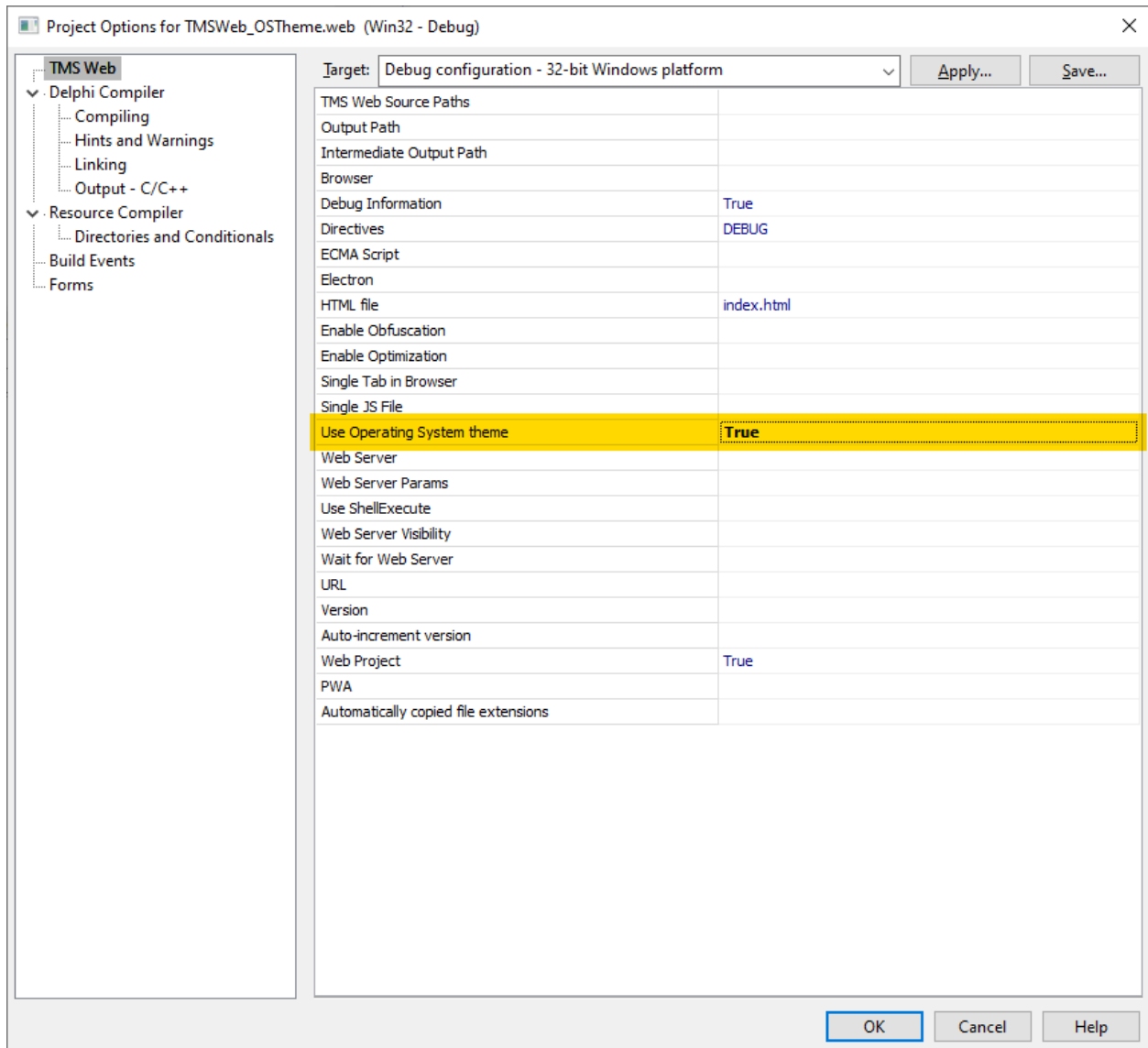
Now, CSS can take care of further styling of the generated HTML elements.

Theming

Meanwhile, all major desktop and mobile operating systems have introduced the concept of a light and dark themes to accommodate the typical preference of dark themes for young computer users and light themes for older computer users. Naturally, there is a tendency that young computer users will come to expect that a web application adopts a dark theme and vice versa for older users. Meanwhile, browsers offer capabilities of detecting whether the operating system where the browser runs is configured for a dark theme or light theme. And so, a TMS WEB Core application can automatically run using a dark theme or a light theme depending on these settings. Of course, this feature is optional, and it can be used in an automatic way or you can add application level code for switching to your desired theme in a customized way.



To enable this feature, go into project options and enable automatic theming via:



BiDiMode

Default TMS WEB Core web client applications are designed for left to right written languages. For languages written from right to left, you can application wide configure the browser to use right to left rendering. To do this, edit the project main HTML file and add the attribute `dir="rtl"` for the `<HTML>` tag:


```
<!DOCTYPE html>
<html dir="rtl" lang="en">
<head>
  <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <meta $(ThemeColor)>
  <noscript>Your browser does not support JavaScript!</noscript>
  <link rel="icon" href="data:;base64,=">
  <meta $(Manifest)>
  <title>TMS Web Project</title>
  <script type="text/javascript" src="$(ProjectName).js"></script>
  <style>
  </style>
</head>
<body>
<meta $(BodyParameters)>
</body>
<script type="text/javascript">
  rtl.run();
</script>
</html>
```

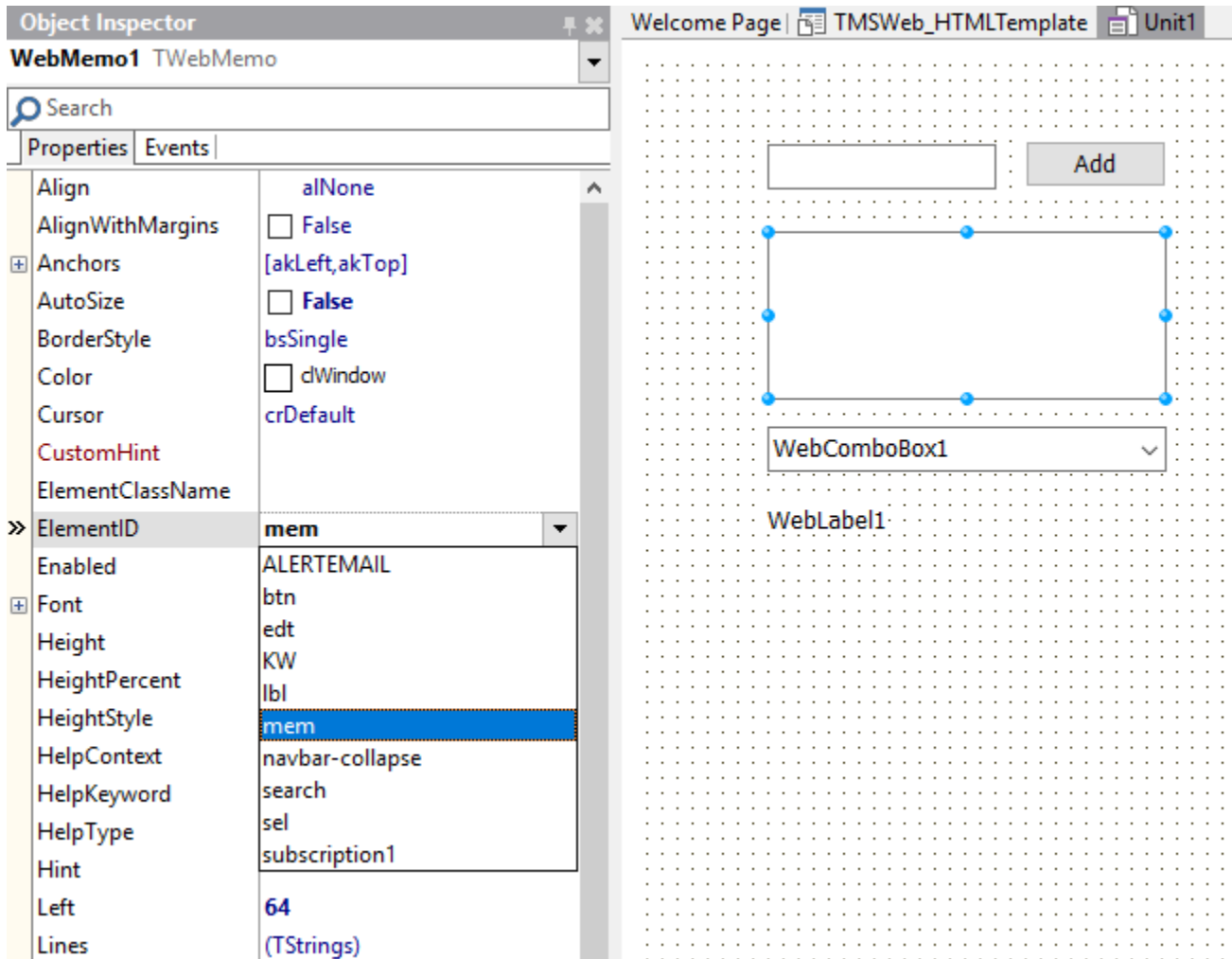
If you only want to enable right-to-left for specific controls on a page that is mainly left-to-right, you can use the control's BiDiMode property for this (similar as in Delphi VCL applications).

Use of HTML templates

The TMS WEB Core framework is also completely open to have the page layout designed directly from HTML & CSS. The architecture of the framework provides for separating design & code and even have the design done by people with a role, i.e. graphical designers.

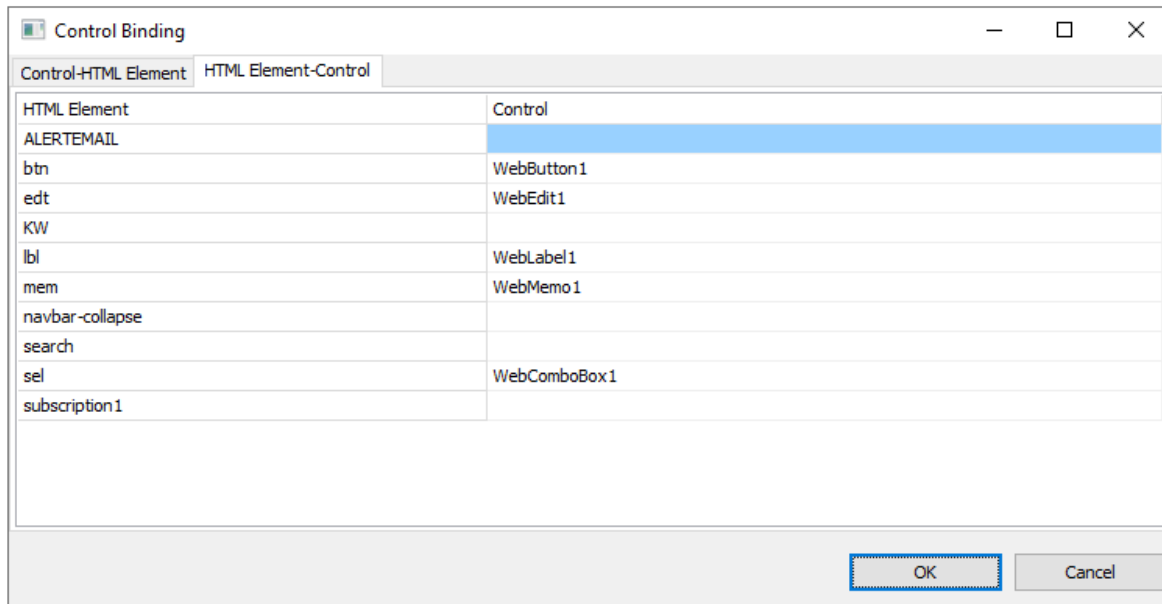
So, how is this separation handled? Fortunately, in a very easy and straightforward way. The link between HTML and the UI controls and code used in the Delphi IDE is based on the unique HTML element ID. Every TMS WEB Core control has a property ElementID. When the ElementID is not used, i.e. left empty, the HTML elements the control consists of is generated by the TMS WEB Core framework. When the ElementID is specified, the HTML element found is hooked up to the Pascal class for the control. This means that property accessors directly get and set values from the HTML element and the various HTML element Javascript events are hooked up the class and exposed as Pascal event handlers.

Here the TWebMemo is hooked up via the ElementID property to a TEXTAREA HTML element with ID set to "mem" and already in the HTML file.



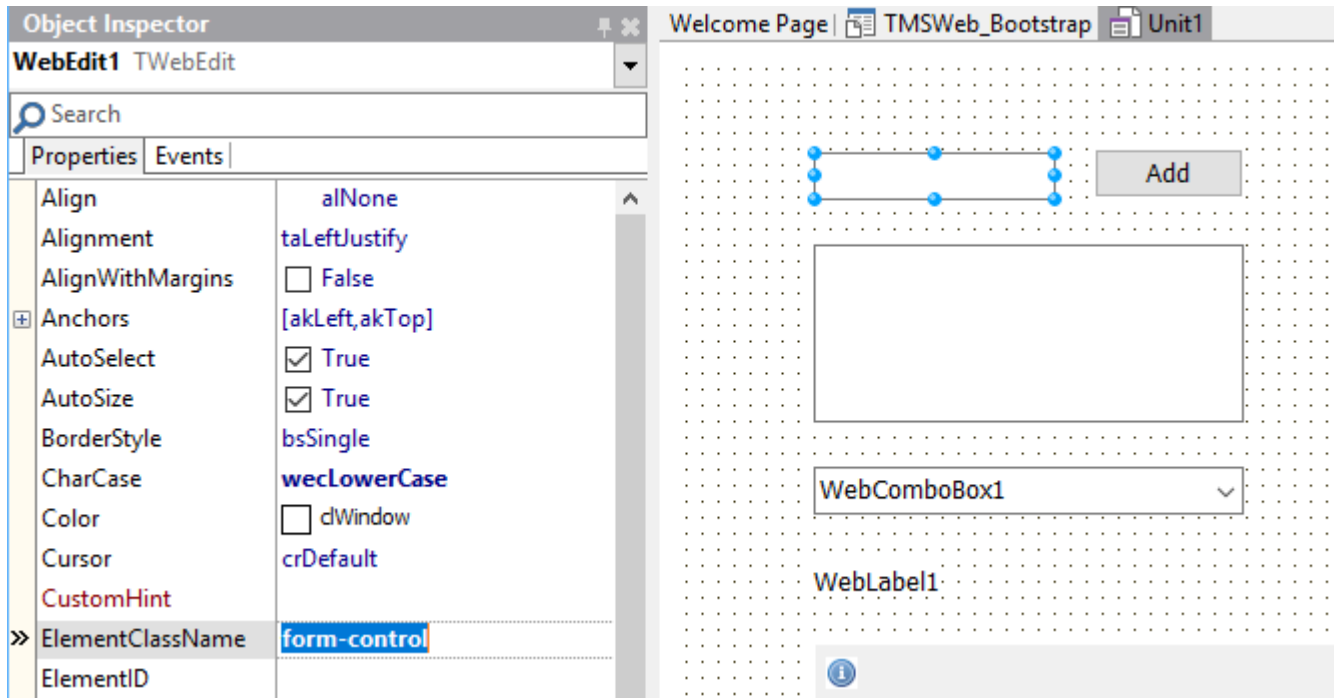
The software developer and the graphical designer can collaborate by simply ensuring that the designer provides the HTML element IDs to the software developer or the software developer can provide a list of IDs of controls needed to the graphical designer.

Alternatively, the mapping between UI controls on the form designer and HTML elements can also be done via the binding editor which is invoked from the TWebForm context menu of the form designer:



In this control binding editor, two views are possible: the view that shows the UI controls found on the form in the first column and the possibility to pick in the right column the HTML element to map the control to and vice versa in the HTML Element-Control tab.

It speaks for itself that using this technique empowers us to take advantage of responsive design for TMS WEB Core web applications. When the HTML template for the page is applying responsive design techniques, i.e. different layouts for different device screen sizes, the UI controls will appear where the designer defined these should appear depending on the screen size. That is not all though. It is also possible to let the Delphi designed UI be generated in the body part of a HTML page or in any specified HTML container element in a HTML page. As such, a graphical designer could create a page layout with a header, footer and other elements in the HTML page and add a specific area via a HTML DIV or SPAN element where the Delphi designed UI will be generated in. To do so, all that is needed is set to the ID for the HTML element where the form should be generated via the `Form.AppContent` property. Finally, each UI control also exposes an `ElementClass` property. Via this `ElementClass` property a CSS style can be specified for an UI control. Via this way for example, it is very easy to use a popular framework like bootstrap. It is sufficient to set the bootstrap CSS class names to the UI controls on the Delphi form designer by their `ElementClass` properties.



Here the ElementClass property of the edit control on the form is set to the bootstrap 'form-control' style:

One of the demos included in the TMS WEB Core framework shows this. By simply changing the bootstrap theme via changing the reference in the HTML page template, the appearance of the web application will adapt automatically.

Demo without styling:

http://www.tmssoftware.biz/tmsweb/demos/tmsweb_simple/

Demo with bootstrap styling applied:

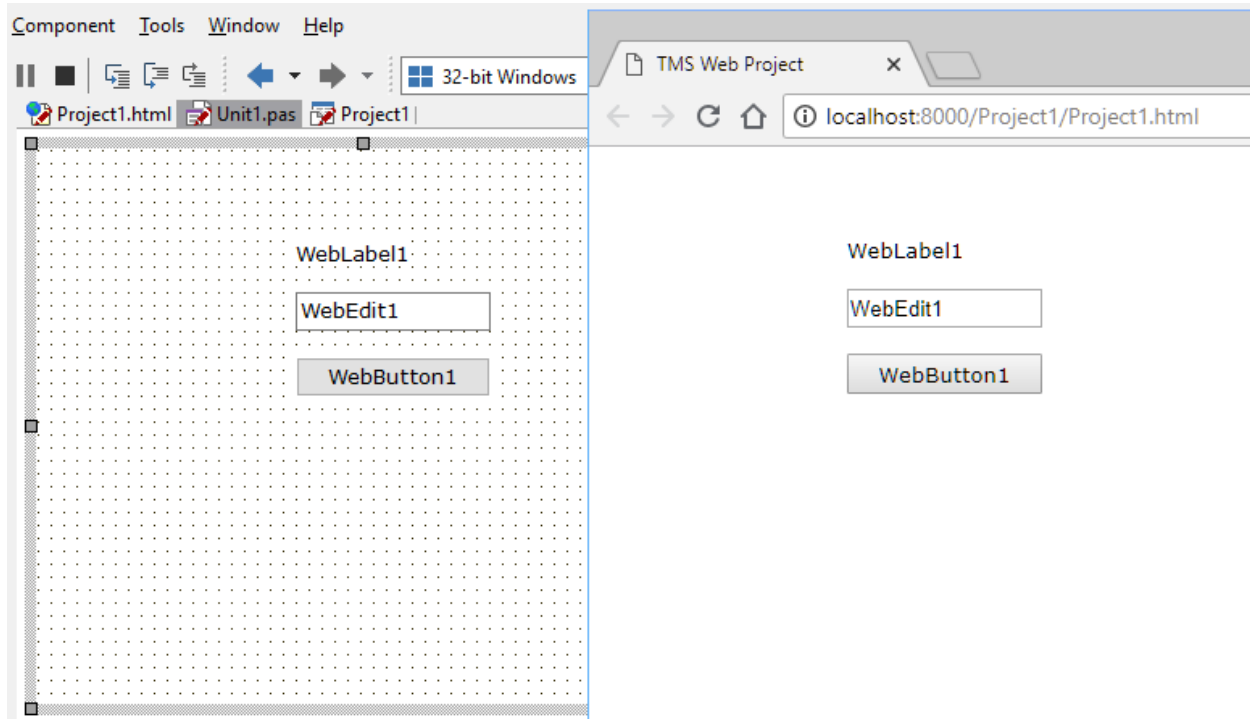
http://www.tmssoftware.biz/tmsweb/demos/tmsweb_bootstrap/

Further fine-tuning on how the design-time setup translates to run-time look & feel and layout is possible via the UI control properties ElementFont and ElementPosition.

Default, the UI control ElementFont property is set to efProperty. This means that the UI control Font property values will be used to generate the style attributes for the HTML element (in case ElementID and ElementClassName are blank). When ElementFont is set to efCSS, this means the font for the HTML element will be determined by the browser CSS resolving.

Example:

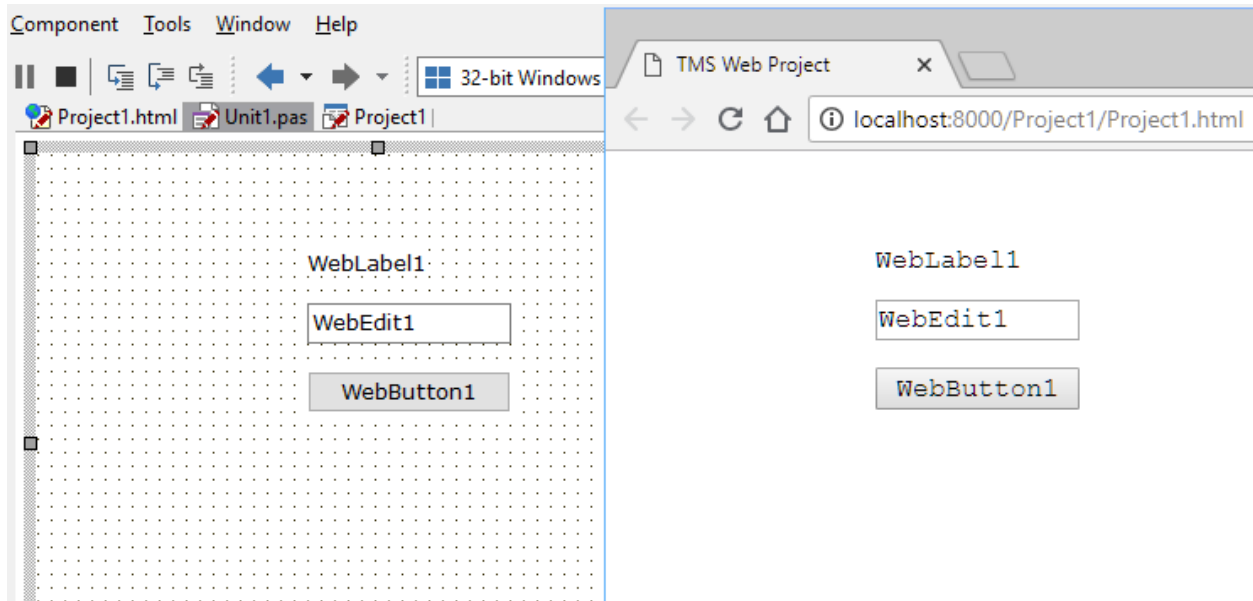
This is a TMS WEB Core project web form with 3 controls. The font for the controls was set at design-time to Verdana, 10pt. In the browser, this renders exactly the same:



Now, changing the ElementFont property on the 3 controls to efCSS and including the following CSS in the form's unit1.html:

```
<html>
<head>
  <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
  <title>TMS Web Project</title>
  <style>
    * {
      font-family: Courier New;
      font-size: 12pt;
    }
  </style>
</head>
<body>
</body>
</html>
```

results in:



The `ElementPosition` property determines how the form designer based coordinates are used as style attributes for the HTML element. When `ElementPosition` is set to `epAbsolute` (default), the HTML element style attributes are set to absolute and the control position and size will match exactly how it was designed in the form designer. When the setting is `epRelative` or `epNone`, the HTML element layout, position and size will be determined by the browser and possible CSS applied to the HTML element(s).

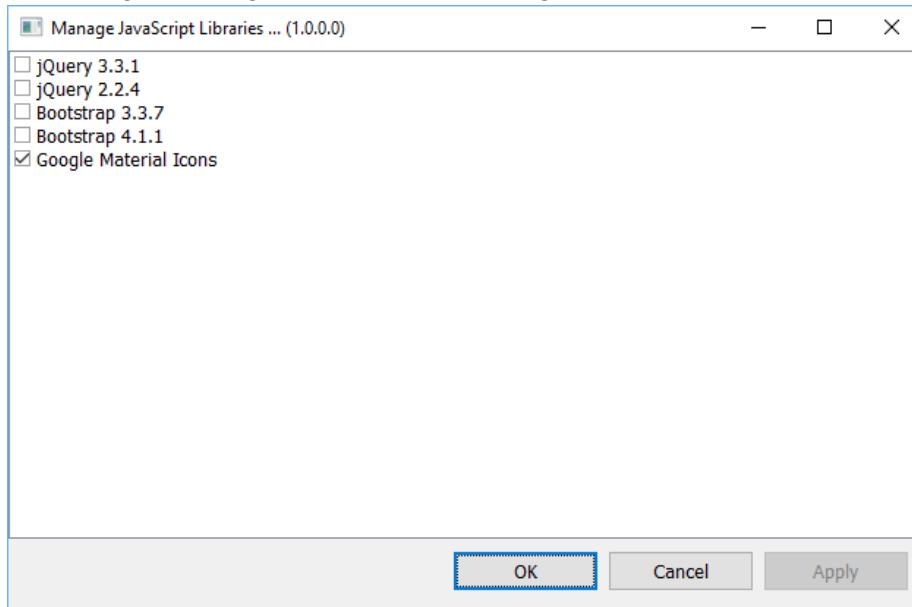
JavaScript and CSS

In the previous paragraph, it was explained how a form uses a HTML file and that the HTML file can contain HTML elements, CSS, JavaScript as well as references to existing JavaScript libraries and CSS. While these references can always be manually added to the HTML file, the IDE also provides for automatic insertion or removal of such references.

To do this, choose from the project context menu in the project explorer pane in the IDE the menu option:

“Manage JavaScript Libraries ...”

This brings a dialog with several preconfigured popular JavaScript libraries that can be added:

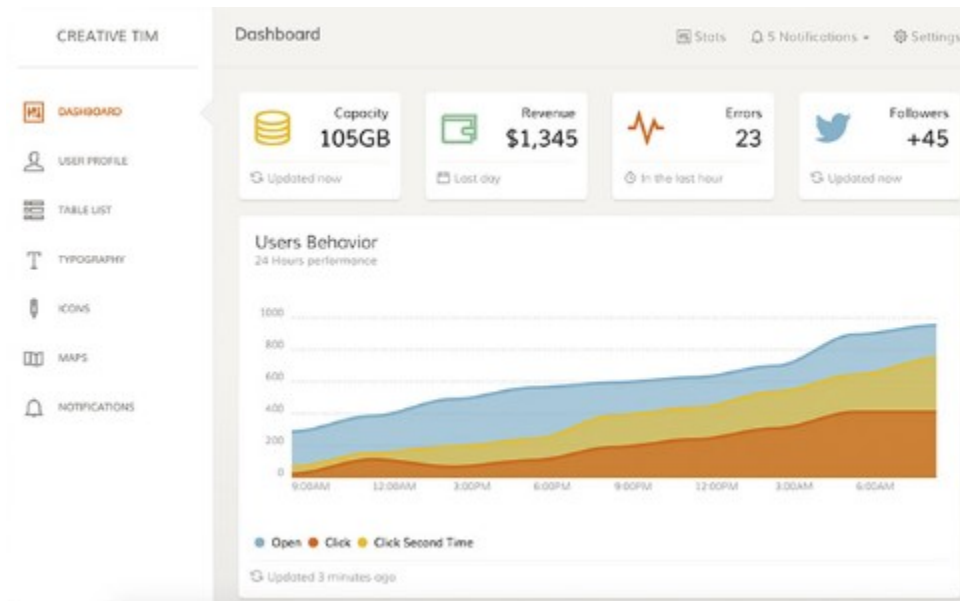


From this dialog, simply check the JavaScript libraries you want to use for your project.

Using off the shelf HTML templates

This chapter explains step by step a typical scenario for adopting an existing 3rd party HTML template for use in your application.

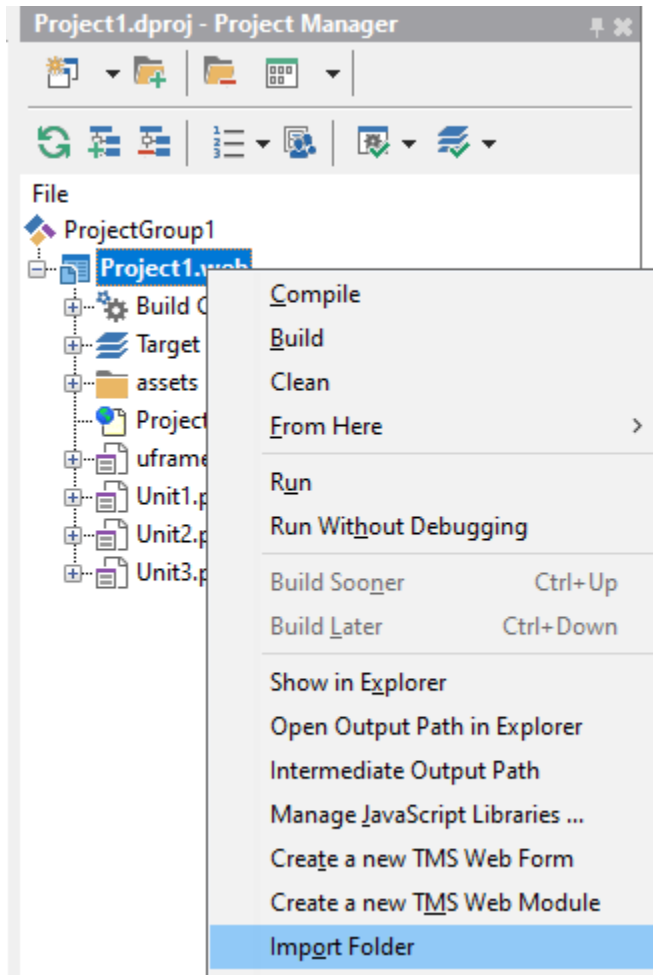
In this example, we will highlight step by step how such 3rd party template can be integrated into a TMS WEB Core web application. For this example, we will use a free off the shelf HTML template as available from <https://www.creative-tim.com/> in particular the Paper Dashboard template <https://www.creative-tim.com/product/paper-dashboard>



This template offers a modern design and is responsive. The sidebar will collapse when the device screen is small.

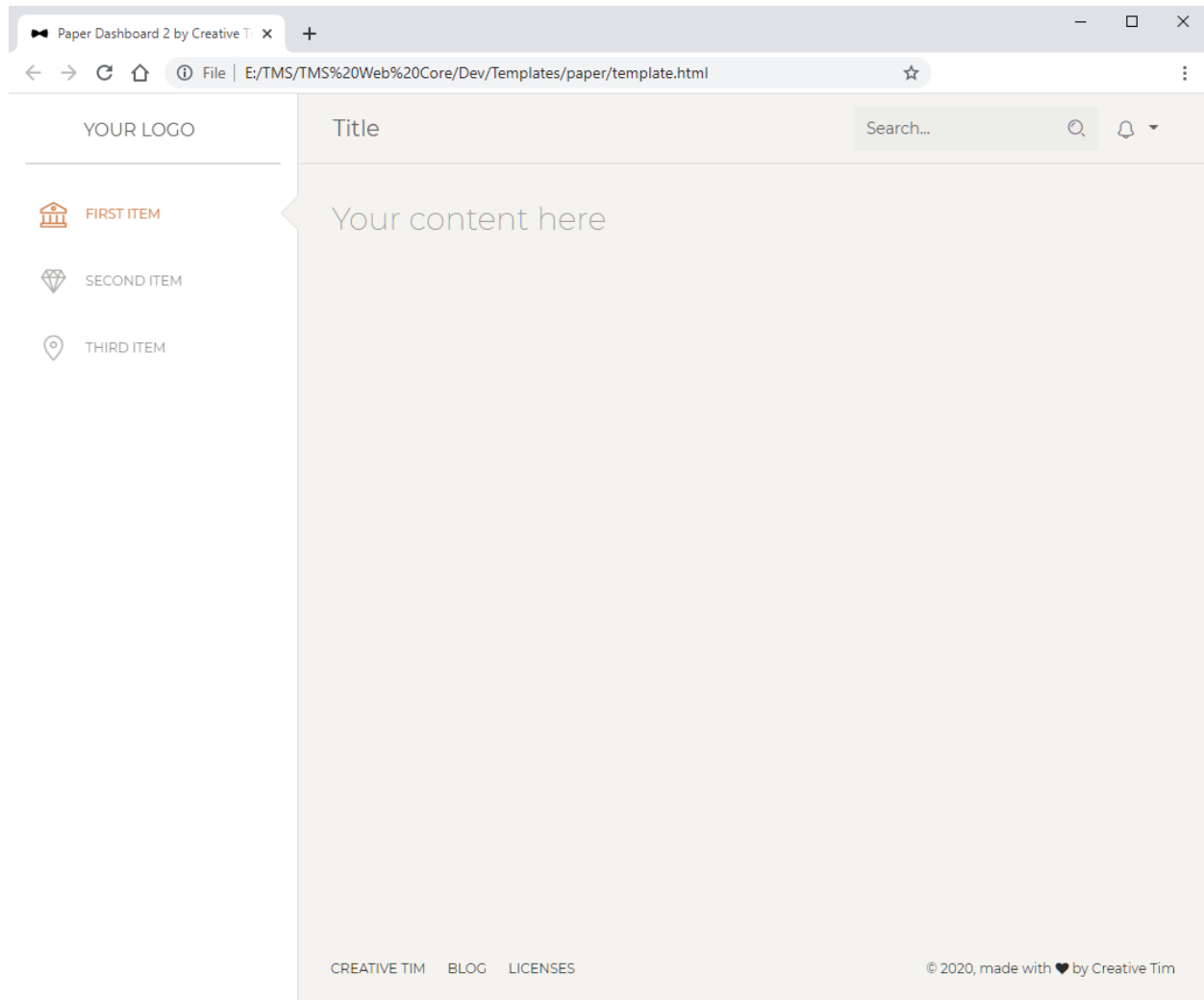
After downloading the template, unzip the distribution and in the main folder of the distribution we see template.html and a folder assets. The assets folder contains all css, images, fonts, JavaScript used in this template.

Copy this assets folder under your project folder and import it into the project from the IDE with the "Import folder" function found in the project context menu:



After importing, all files under the assets folder are added to the project and will as such also be automatically deployed when running the application.

Next we look at template.html. This looks like:



It is this template we are going to use for a form in the TMS WEB Core application. As explained in the previous chapters, each form in a TMS WEB Core application has associated HTML. It is for this form HTML we are going to use the template. Note that the form's HTML is loaded dynamically when the application loads the form. The application itself is started from the project HTML file.

To start using the template for a website for a TMS WEB Core application form, the first recommended step is to look into the template HTML for references to external JavaScript or CSS libraries. Moving these references to the main project HTML file has the advantage that all the libraries are already loaded by the browser when the form is being loaded.

For this template, following library references are used:

```
<!-- Core JS Files -->
<script src="./assets/js/core/jquery.min.js"></script>
<script src="./assets/js/core/popper.min.js"></script>
<script src="./assets/js/core/bootstrap.min.js"></script>
<script src="./assets/js/plugins/perfect-scrollbar.jquery.min.js"></script>
<!-- Google Maps Plugin -->
<script src="https://maps.googleapis.com/maps/api/js?key=YOUR_KEY_HERE"></script>
<!-- Chart JS -->
<script src="./assets/js/plugins/chartjs.min.js"></script>
<!-- Notifications Plugin -->
<script src="./assets/js/plugins/bootstrap-notify.js"></script>
<!-- Control Center for Now Ui Dashboard: parallax effects, scripts for the example pages etc -->
<script src="./assets/js/paper-dashboard.min.js?v=2.0.1" type="text/javascript"></script>
```

We won't use Google Maps or charts in the demo using the template, so these can be removed. All other references are cut from the template HTML file and pasted into the project HTML file.

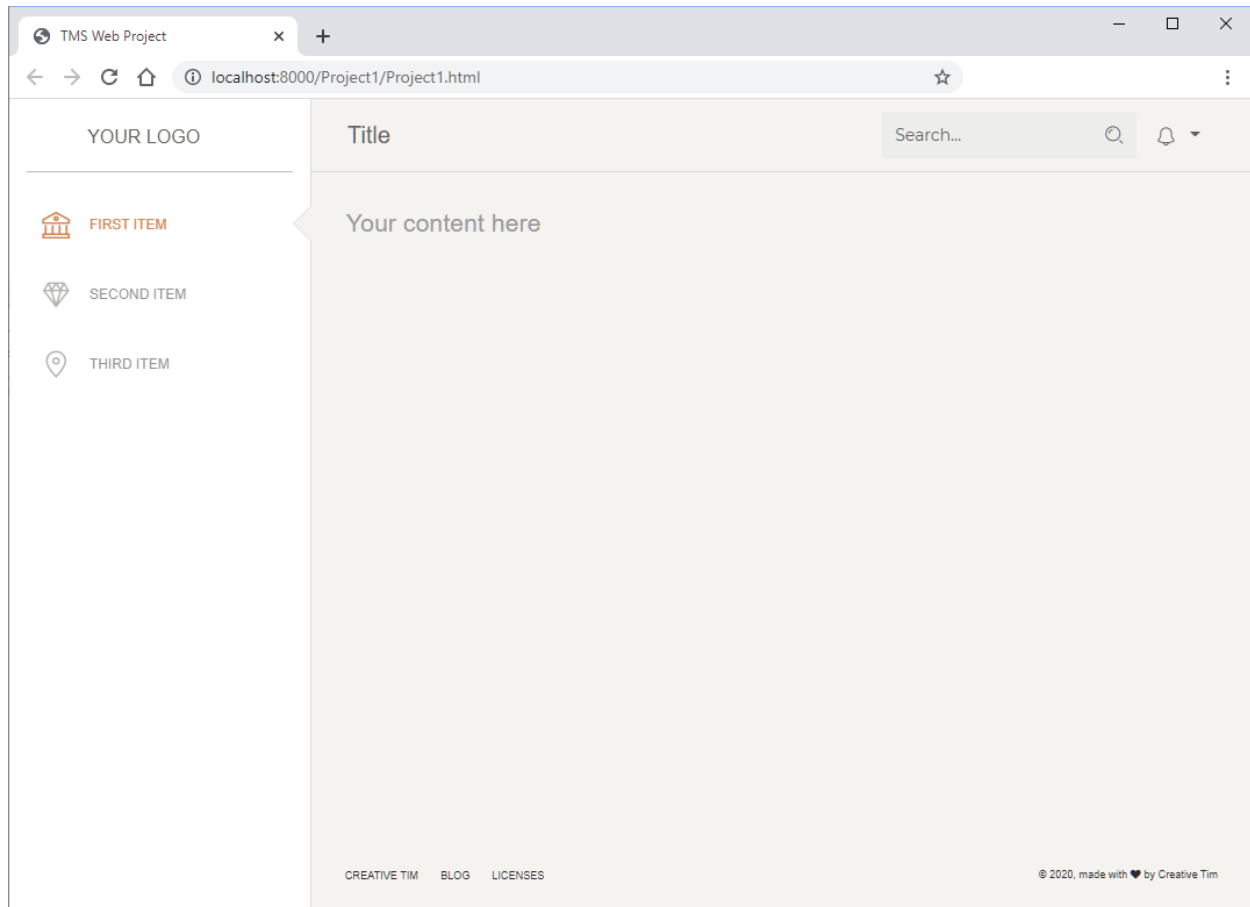
This way, the project HTML file becomes (in highlight the library references added):

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <meta $(ThemeColor)>
    <noscript>Your browser does not support JavaScript!</noscript>
    <link rel="icon" href="data:;base64,=">
    <meta $(Manifest)>
    <title>TMS Web Project</title>

    <!-- Core JS Files -->
    <script src="./assets/js/core/jquery.min.js"></script>
    <script src="./assets/js/core/popper.min.js"></script>
    <script src="./assets/js/core/bootstrap.min.js"></script>
    <script src="./assets/js/plugins/perfect-scrollbar.jquery.min.js"></script>
    <!-- Notifications Plugin -->
    <script src="./assets/js/plugins/bootstrap-notify.js"></script>
    <!-- Control Center for Now Ui Dashboard: parallax effects, scripts for the example pages etc -->
    <script src="./assets/js/paper-dashboard.min.js?v=2.0.1" type="text/javascript"></script>

    <script type="text/javascript" src="$(ProjectName).js"></script>
    <style>
    </style>
  </head>
  <body>
    <meta $(BodyParameters)>
    </body>
    <script type="text/javascript">
      rtl.run();
    </script>
  </html>
```

Normally, we could copy the remaining HTML from the template.html file into the form's unit1.html now and we will have the first TMS WEB Core web application based on this HTML template:



Next step will be to couple the TMS WEB Core web application to the template. The first items we will couple are the left sidebar items and the page title. The content of the TMS WEB Core form will be displayed in the content area of the template.

Sidebar

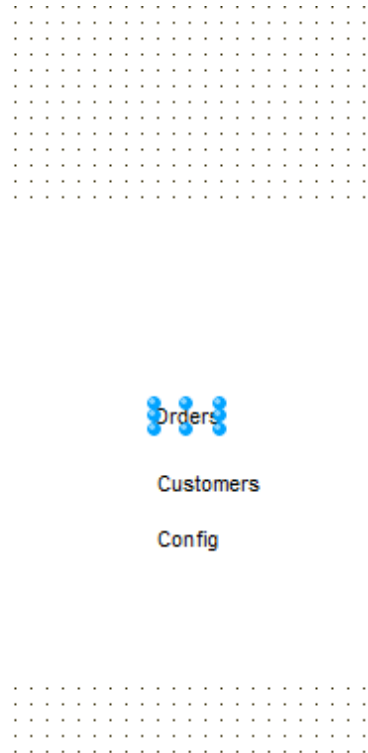
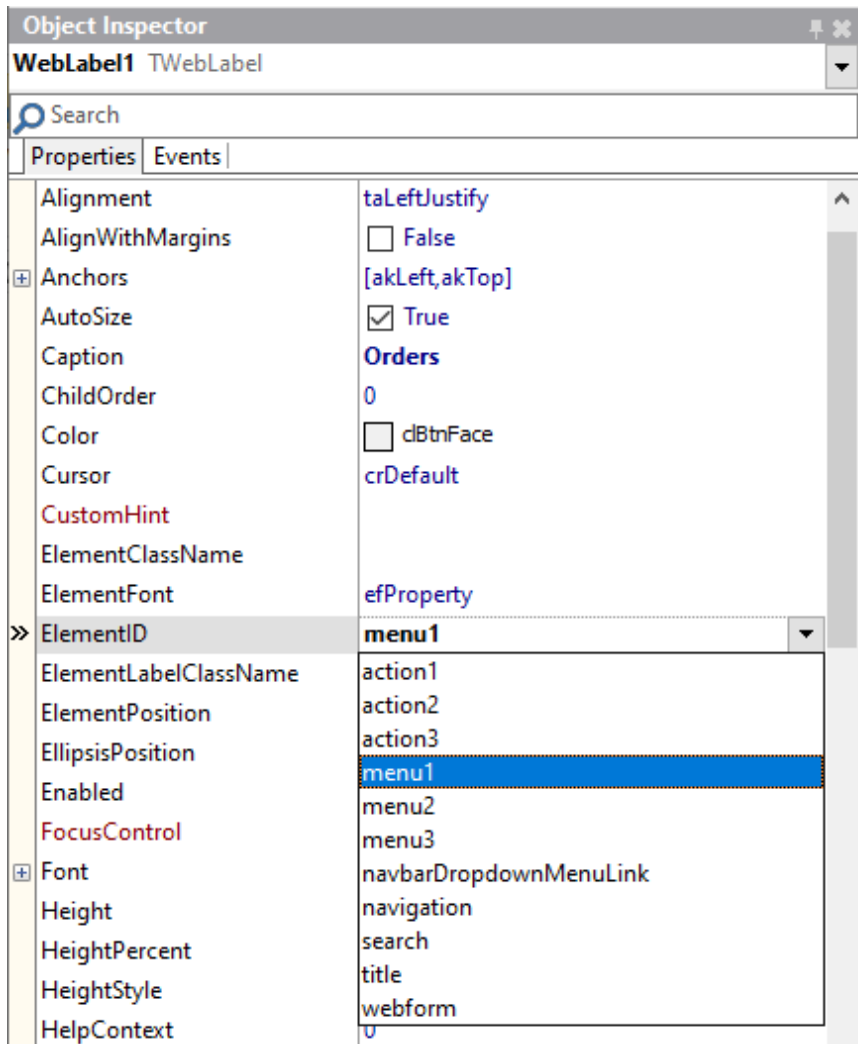
We locate in the HTML file the 3 sidebar items.

```
<div class="sidebar-wrapper" style="overflow:hidden">
  <ul class="nav">
    <li class="active">
      <a href="javascript:;">
        <i class="nc-icon nc-bank"></i>
        <p>First Item</p>
      </a>
    </li>
    <li>
      <a href="javascript:;">
        <i class="nc-icon nc-diamond"></i>
        <p>Second Item</p>
      </a>
    </li>
    <li>
      <a href="javascript:;">
        <i class="nc-icon nc-pin-3"></i>
        <p>Third Item</p>
      </a>
    </li>
  </ul>
</div>
```

and add a unique element ID to these <p> HTML elements for the sidebar items. Adding IDs "menu1", "menu2", "menu3", this becomes:

```
<div class="sidebar-wrapper" style="overflow:hidden">
  <ul class="nav">
    <li class="active">
      <a href="javascript:;">
        <i class="nc-icon nc-bank"></i>
        <p id="menu1">First Item</p>
      </a>
    </li>
    <li>
      <a href="javascript:;">
        <i class="nc-icon nc-diamond"></i>
        <p id="menu2">Second Item</p>
      </a>
    </li>
    <li>
      <a href="javascript:;">
        <i class="nc-icon nc-pin-3"></i>
        <p id="menu3">Third Item</p>
      </a>
    </li>
  </ul>
</div>
```

Now, we can add 3 TWebLabel components on the TMS WEB Core form and connect these labels to the <p> elements. This is done via the WebLabel.ElementID property.

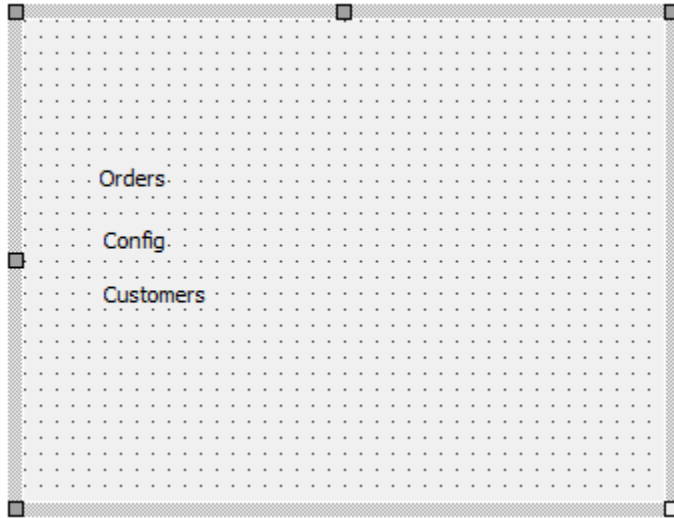


Note that the TMS WEB Core plugin automatically detects all HTML elements where an ID is set and displays these in the dropdown for ElementID property. Further, we set for the 3 added labels ElementFont = efCSS and WidthStyle, HeightStyle to ssAuto. It will be the template that controls this label font and label size. We set the label captions to “Orders”, “Customers”, “Config” respectively as these will be the sidebar items.

Next, we do the same for the “Title” label in the template and connect it to over the ElementID property to another label on the form. This allows us to set per form a title controlled from the TMS WEB Core application code.

We also want to control where the TMS WEB Core form will be displayed within this template.

As the goal is to bind 3 forms to this template, the most efficient way will be to add a frame with these 3 sidebar labels that will be reused on each form.

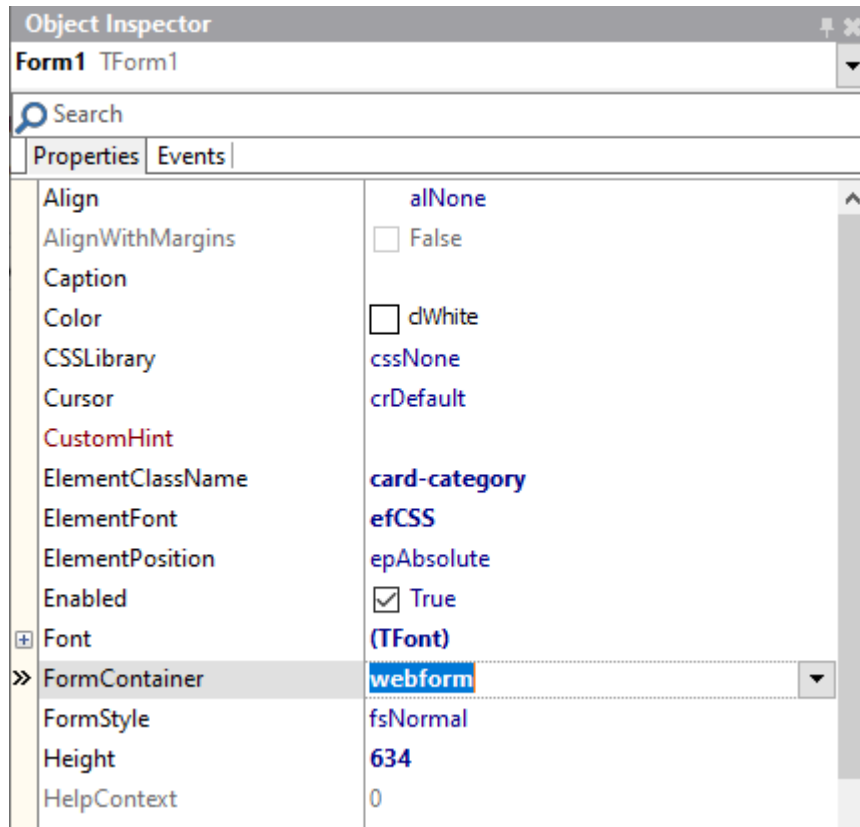


In the template we find the content area under the navbar:

```
<!-- End Navbar -->
<div class="content">
  <div class="row">
    <div class="col-md-12">
      <h3 class="description">Your content here</h3>
    </div>
  </div>
</div>
```

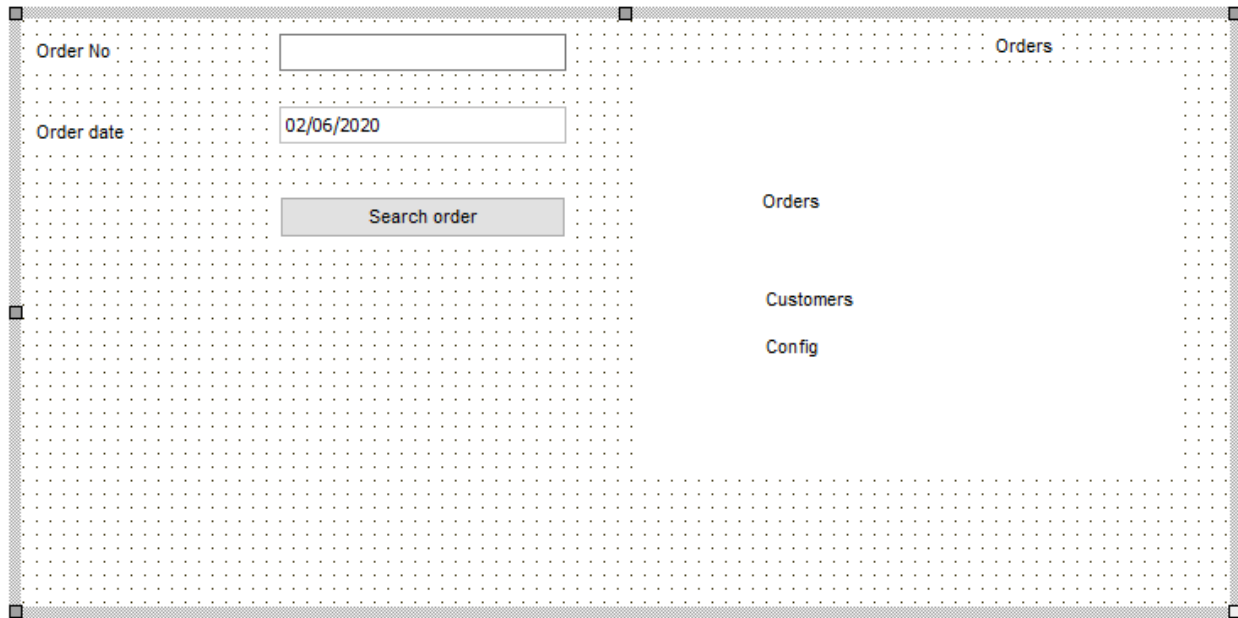
Also here, we introduce a unique ID to the element where we want the form to be rendered. The ID is “webform” and to make a form appear within this HTML element, all we need to do is set the property WebForm.FormContainer to this ID.

```
<div class="content">
  <div class="row">
    <div class="col-md-12">
      <h3 id="webform" class="description"></h3>
    </div>
  </div>
</div>
```

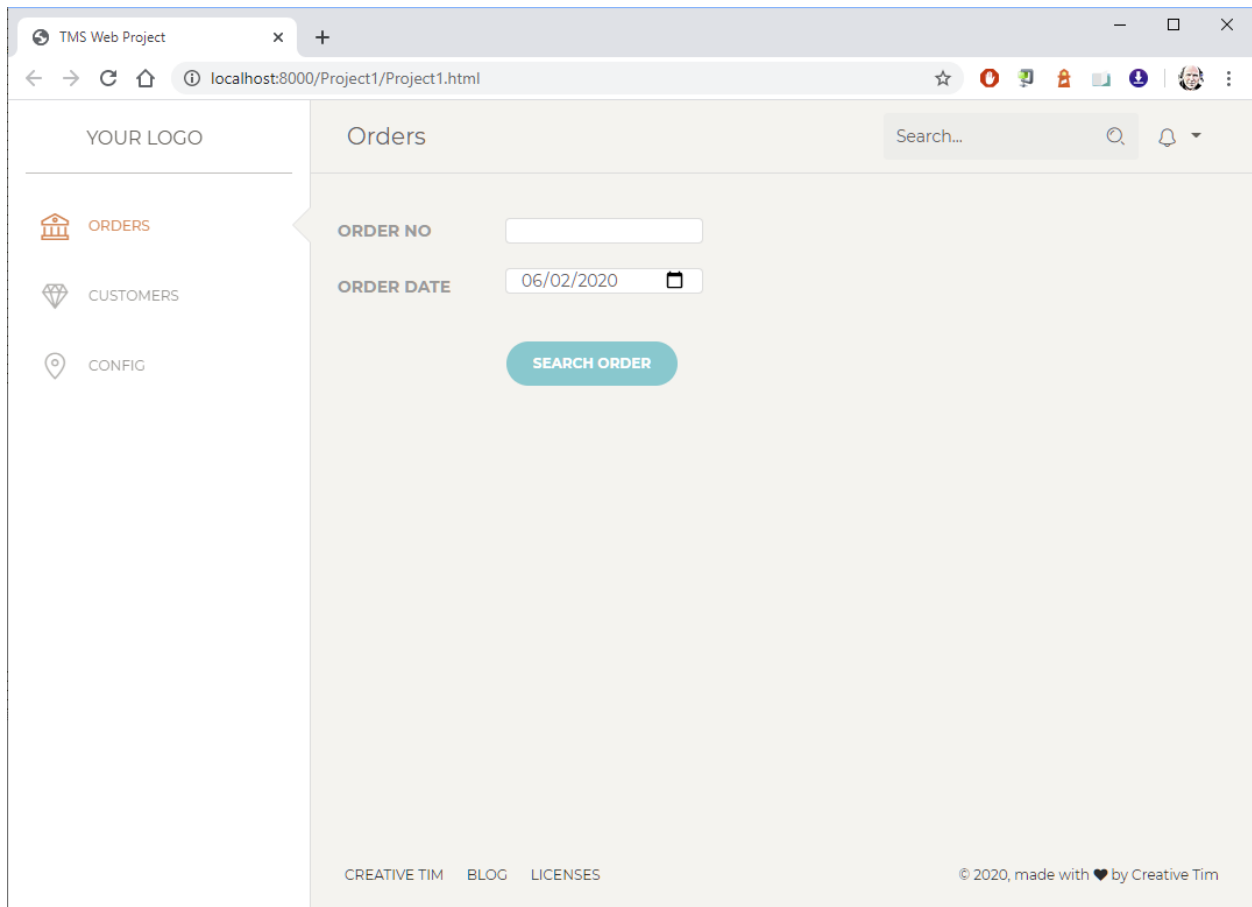


Note that the form's ElementFont was set to efCSS to pickup the CSS that applies to the element where the form will be rendered. We also set the ElementClassName to a CSS class defined by the template so the font of controls will match the font used in the template.

Let's drop some controls on the TMS WEB Core form and see how the result looks in the browser:



becomes:



Note that for the edit & datepicker controls, the `ElementClassName` was set to “form-control”, a bootstrap style. The button control `ElementClassName` was set to: “btn btn-primary btn-round” to show this green rounded shape.

Now it is time to add the code that will take care of loading a different TMS WEB Core form when a sidebar item is clicked. Therefore, add `OnClick` event handlers for the labels on the frame that are connected to the sidebar HTML elements.

We create one wrapper function that can load the form by just passing the form class.

```
procedure TMenuFrame.LaunchForm(AInstanceClass: TFormClass);
var
    frm: TForm;

    procedure FormCreated(AForm: TObject);
    begin
        (AForm as TForm).Show;
    end;

begin
    if Uppercase(Application.ActiveForm.ClassName) <>
    Uppercase(AInstanceClass.ClassName) then
    begin
        Application.CreateForm(AInstanceClass, 'body', frm, @FormCreated);
    end;
end;
```

Note that the `FormCreated` method is asynchronously loaded when the form HTML was loaded. In a browser, loading such external form HTML file is always an asynchronous process.

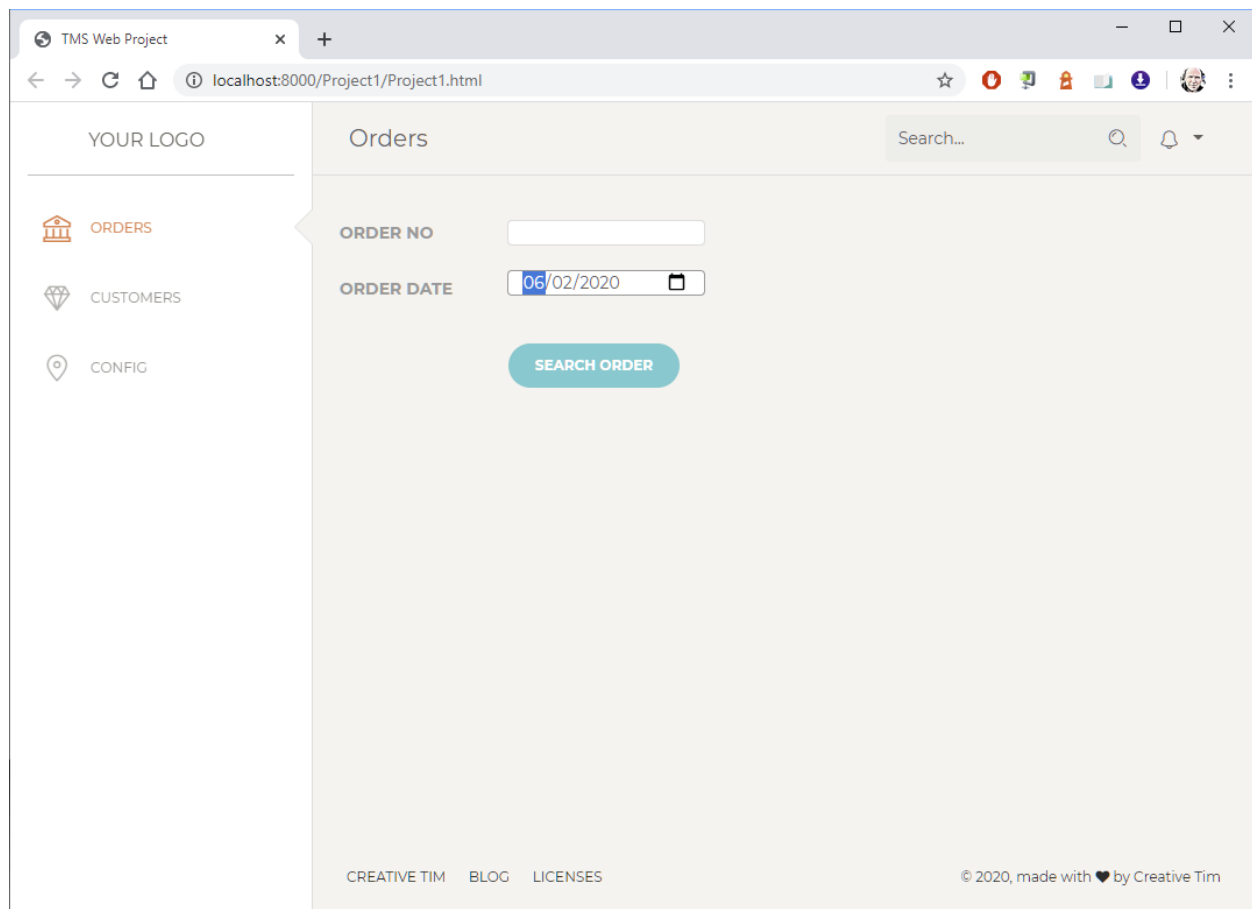
This way, the click handlers for the sidebar labels become simply:

```
procedure TMenuFrame.WebLabel1Click(Sender: TObject);
begin
    LaunchForm(TForm1);
end;

procedure TMenuFrame.WebLabel2Click(Sender: TObject);
begin
    LaunchForm(TForm2);
end;
```

```
procedure TMenuFrame.WebLabel3Click(Sender: TObject);
begin
    LaunchForm(TForm3);
end;
```

The end result becomes:



There is one more detail here handled in this example at template level. As we added multiple forms to the project, the template for each form in this project will be the same for one detail and that is the class for the selected item in the sidebar. Note there is a triangle indicating the selected item and the selected item is shown in orange.

So, to move the selected sidebar item to another item when a different form is loaded, the `class="active"` attribute will be moved to the respective item for each of the form's HTML templates:

```
<div class="sidebar-wrapper" style="overflow:hidden">
  <ul class="nav">
    <li class="active">
      <a href="javascript:;">
        <i class="nc-icon nc-bank"></i>
        <p id="menu1">First Item</p>
      </a>
    </li>
    <li>
      <a href="javascript:;">
        <i class="nc-icon nc-diamond"></i>
        <p id="menu2">Second Item</p>
      </a>
    </li>
    <li>
      <a href="javascript:;">
        <i class="nc-icon nc-pin-3"></i>
        <p id="menu3">Third Item</p>
      </a>
    </li>
  </ul>
</div>
```

Or alternatively, we could also do this in code. The advantage of doing this in code is that we could this way keep the HTML template for the 3 forms in the applications identical. This means that whenever a designer wants to modify the page, by changing one template file, all forms in the application will be updated.

For this approach, all we need to do is add element IDs to the sidebar navigation elements and then programmatically set the CSS class for the active element.

```
<div class="sidebar-wrapper" style="overflow:hidden">
  <ul class="nav">
    <li id="side1">
      <a href="javascript:;">
        <i class="nc-icon nc-bank"></i>
        <p id="menu1">First Item</p>
      </a>
    </li>
    <li id="side2">
      <a href="javascript:;">
        <i class="nc-icon nc-diamond"></i>
        <p id="menu2">Second Item</p>
      </a>
    </li>
    <li id="side3">
      <a href="javascript:;">
        <i class="nc-icon nc-pin-3"></i>
        <p id="menu3">Third Item</p>
      </a>
    </li>
  </ul>
</div>
```

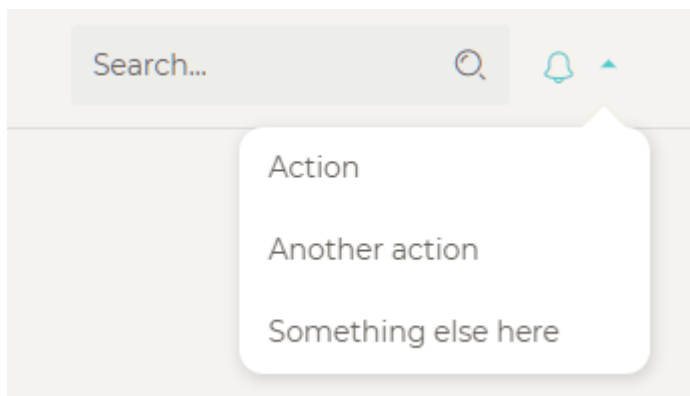
With this approach all we need to do in each form's code is:

```
procedure TForm1.WebFormShow(Sender: TObject);
var
  el: TJSElement;
begin
  // set sidebar element active style
  el := document.getElementById('side1');
  el['class'] := 'active';
end;

procedure TForm2.WebFormShow(Sender: TObject);
var
  el: TJSElement;
begin
  // set sidebar element active style
  el := document.getElementById('side2');
  el['class'] := 'active';
end;

...
```

As you can see in the template file, it has a few other central items. In the top right corner, there is a search bar and a dropdown menu.



This search function and dropdown menu will return for all forms displayed in the content area of the template. So, ideally, this is handled in a centralized way. In TMS WEB Core, we can do this by means of a frame and a `TWebElementActionList`. As the frame is reused on all 3 forms in the application, the event handlers for the elements of the dropdown menu and the search

button can be handled and centralized by the TWebElementActionList on the frame. So, first of all, set unique ID values for the dropdown menu item HTML elements and the search button HTML element. Here we will use “action1”, “action2”, “action3” and “search”.

The dropdown menu is easily recognized in the HTML template and the IDs set:

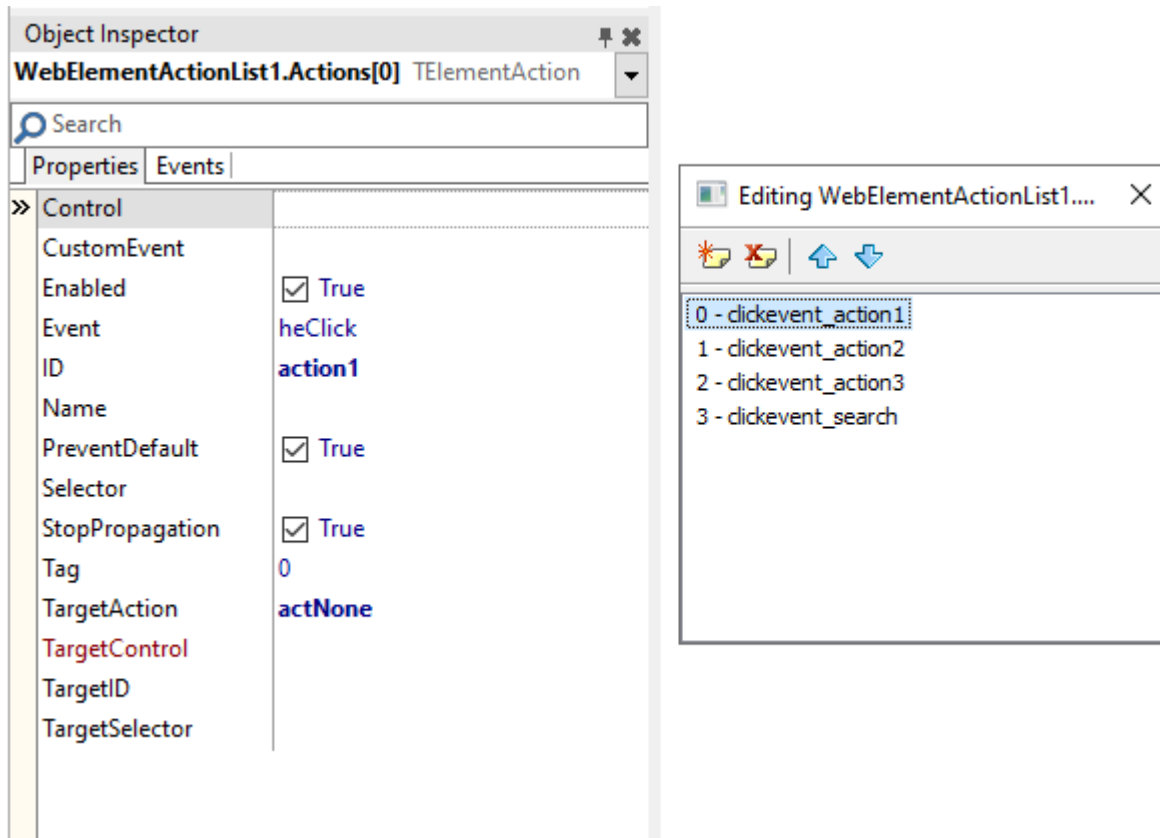
```
<div class="dropdown-menu dropdown-menu-right" aria-labelledby="navbarDropdownMenuLink">
  <a id="action1" class="dropdown-item" href="#">Action</a>
  <a id="action2" class="dropdown-item" href="#">Another action</a>
  <a id="action3" class="dropdown-item" href="#">Something else here</a>
</div>
```

The search button is a material icon found here:

```
<form>
  <div class="input-group no-border">
    <input type="text" value="" class="form-control" placeholder="Search...">
    <div class="input-group-append">
      <div class="input-group-text">
        <i id="search" class="nc-icon nc-zoom-split"></i>
      </div>
    </div>
  </div>
</form>
```

Now, we add a TWebElementActionList on the frame and add 4 actions to handle each of the clicks on the elements given an ID.

Here is the list with 4 items and the ID of the HTML element to handle the click is set to “action1”.



Finally, the OnExecute event handler code is written for the TElementActionList that will handle the events of each of the 4 actions added:

```
procedure TMenuFrame.WebElementActionList1Execute(Sender: TObject;
  AAction: TElementAction; Element: TJSHTMLRecord;
  Event: TJSEventParameter);
begin
  case AAction.Index of
    0: ShowMessage('action 1');
    1: ShowMessage('action 2');
    2: ShowMessage('action 1');
    3: ShowMessage('search');
  end;
end;
```

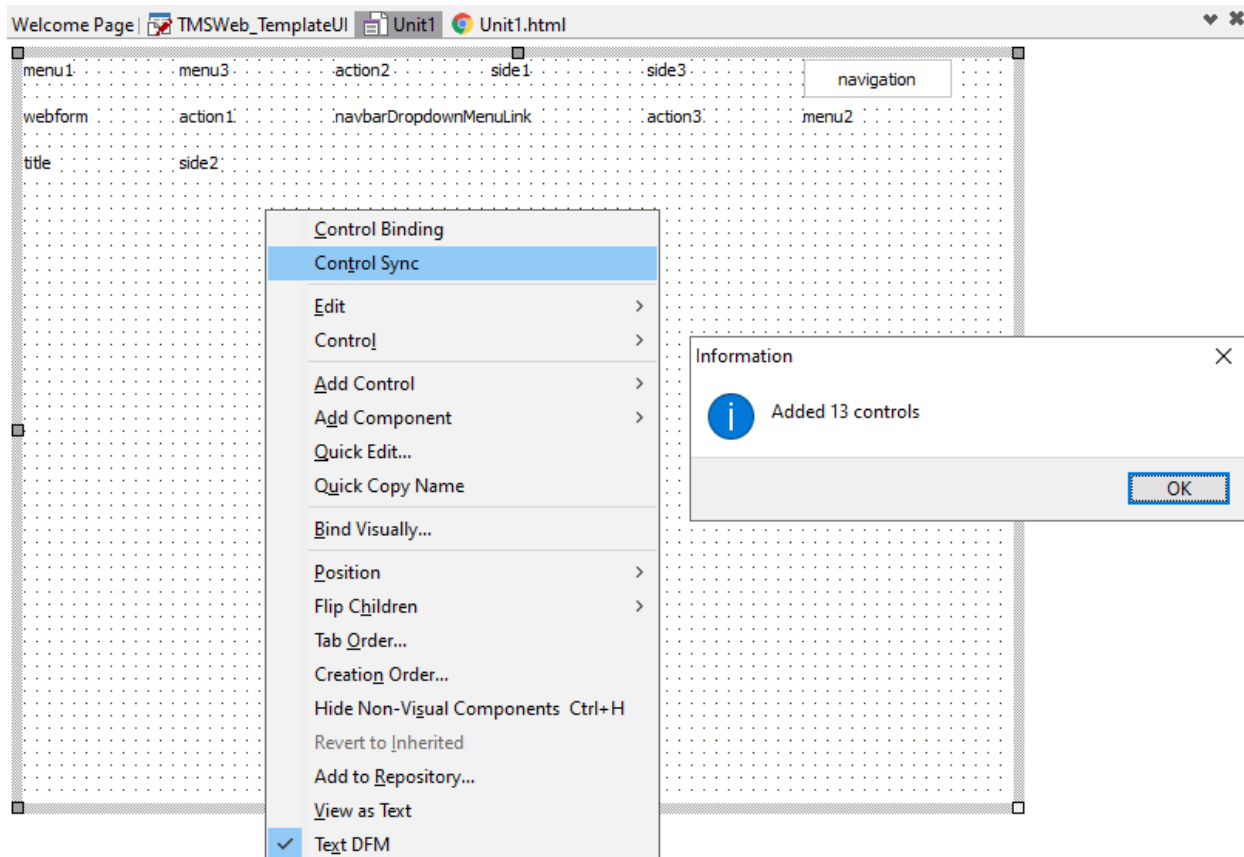
And this completes the HTML template based 3 form TMS WEB Core application.

Automatic synchronisation with HTML templates

It is not necessary to manually put UI controls on the designer for binding to HTML elements in the template. The form designer offers the capability to automatically insert UI controls on the form designer matching with the type of HTML elements in the HTML template.

This can be invoked from the form designer context menu item "Control Sync".

So, all you need to do is add the HTML for your form to the form's HTML file and choose "HTML Sync" from the form's context menu. This will parse the HTML, and it will create an appropriate UI control on the form designer and bind it to the HTML element when it has an ID attribute value. When you change the template later, for example, add more HTML elements, you can do the "HTML Sync" again and the added corresponding UI controls will be added to the form designer. We have a fixed mapping for specific HTML elements to UI controls as well as steered sync by specifying the UI control's class name as the attribute for the HTML element.



The automatic mapping of HTML elements to TMS WEB Core UI controls is based on the following relationship:

HTML Element	UI control class
--------------	------------------

<LABEL>	TWebLabel
<INPUT type="TEXT">	TWebEdit
<INPUT type="NUMBER">	TWebSpinEdit
<INPUT type="CHECK">	TWebCheckBox
<INPUT type="RADIO">	TWebRadioButton
<INPUT type="COLOR">	TWebColorPicker
<INPUT type="DATE">	TWebDateTimePicker
<INPUT type="RANGE">	TWebTrackbar
<SELECT>	TWebComboBox
<TEXTAREA>	TWebMemo
<PROGRESS>	TWebProgressbar
	TWebListControl
	TWebListControl
<BUTTON>	TWebButton
<DIV>	TWebHTMLDiv
	TWebHTMLSpan

In addition to this automatic mapping, it is possible to steer the mapping of the HTML element to a specific UI control with the `twc` attribute.

For HTML elements that have the “`twc`” attribute and an ID, the following mapping happens upon import:

`<ELEMENT twc="classname" id="xx">` → create a new control from class of type `classname`

So, the `twc` attribute has priority to determine the classname of the generated control.

Example:

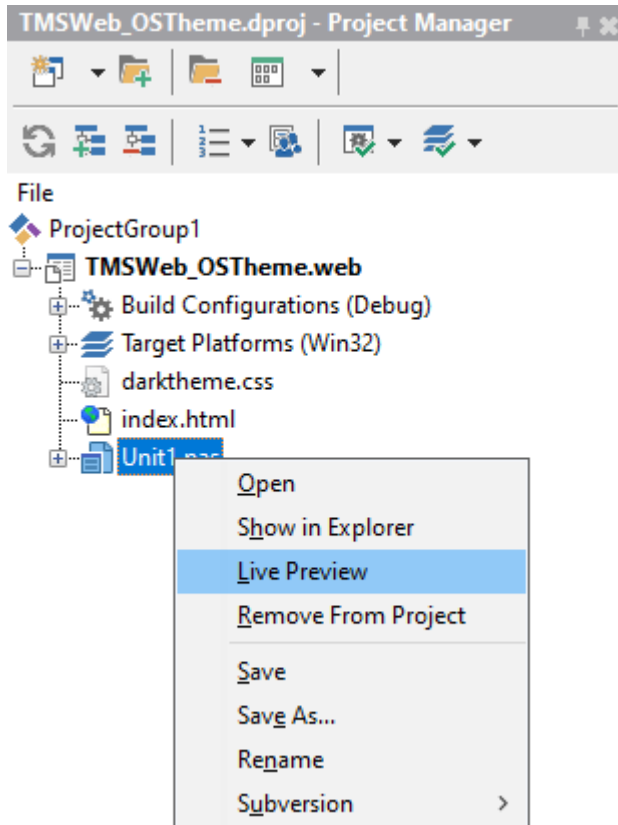
`<DIV ID="mygrid" twc="TWebPanel">` will cause a `TWebPanel` class to be bound to this HTML DIV element on the form designer.

Note that the `twc` attribute can also be used to exclude a HTML element with the ID set to be bound to a UI control when the `twc` attribute is set to “`none`”.

Live preview

Note that the form designer in the Delphi IDE (and also the Lazarus IDE) is based on the VCL (LCL) framework. This means that at design-time, the controls on the form designer are rendered as VCL/LCL controls. While the designer is fast, familiar, and flexible, it is still a different way of rendering it than a real web browser-based rendering. Live preview is a function that allows you to view directly in a separate browser window a live rendering of the form open in the Delphi form designer.

To start a live preview for a specific form in the project, click the form's unit in the Delphi IDE project manager and select "Live Preview".



This will bring up a browser (the browser is the default browser or a specific selected browser from the TMS WEB Core toolbar in the IDE).

Once the live preview is ready and shows the selected form rendered in the browser:

- Make a change to a control's property,

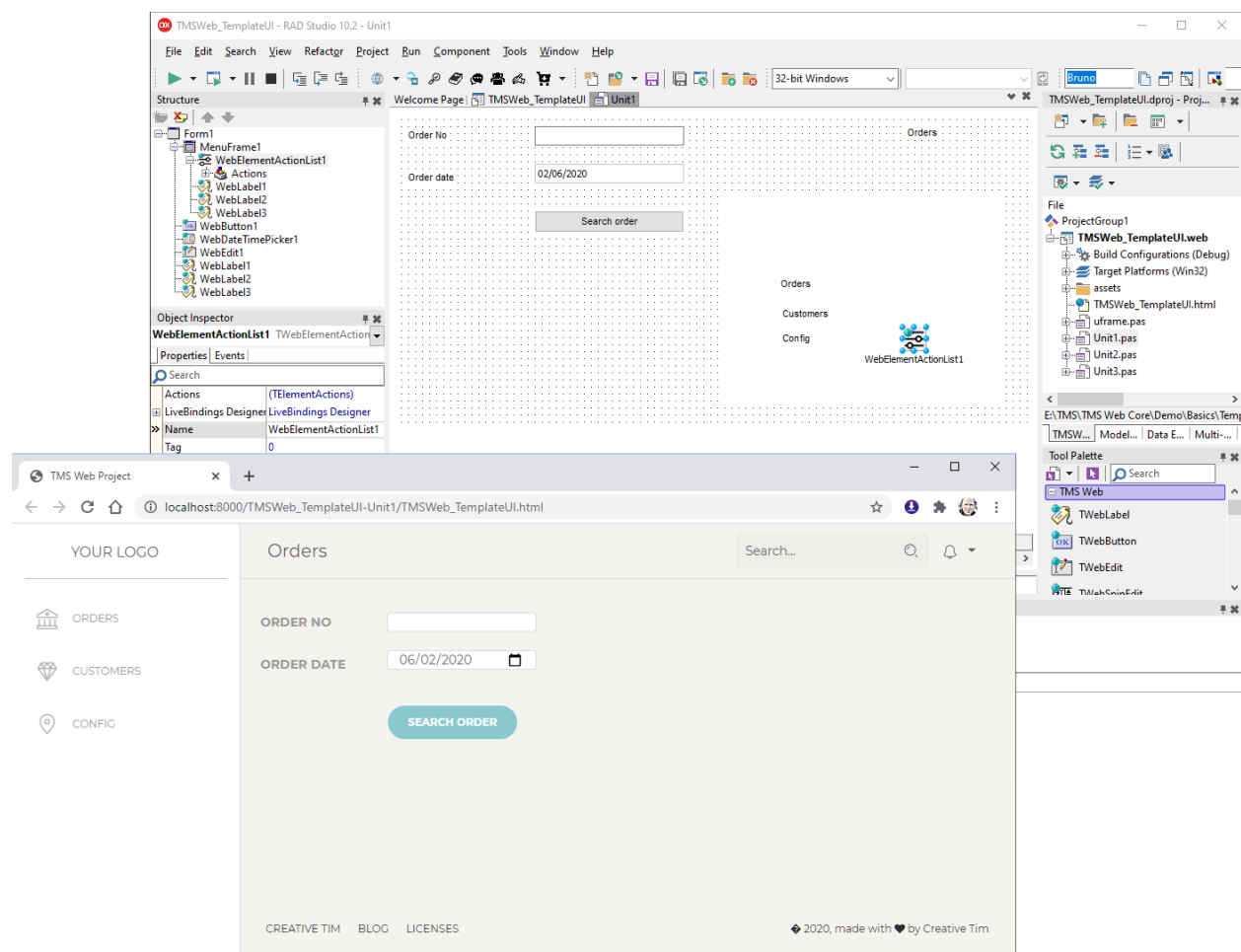
or

- Add a new control on the form or change something in the layout

and

- Press Ctrl-S.

This triggers the live preview browser window to automatically update. The trick here is that live, a single form project consisting of this form in the designer is compiled live and shown in the web browser. Evidently, also when you use HTML templates, the live preview takes this into account and gives you a real preview of this form.



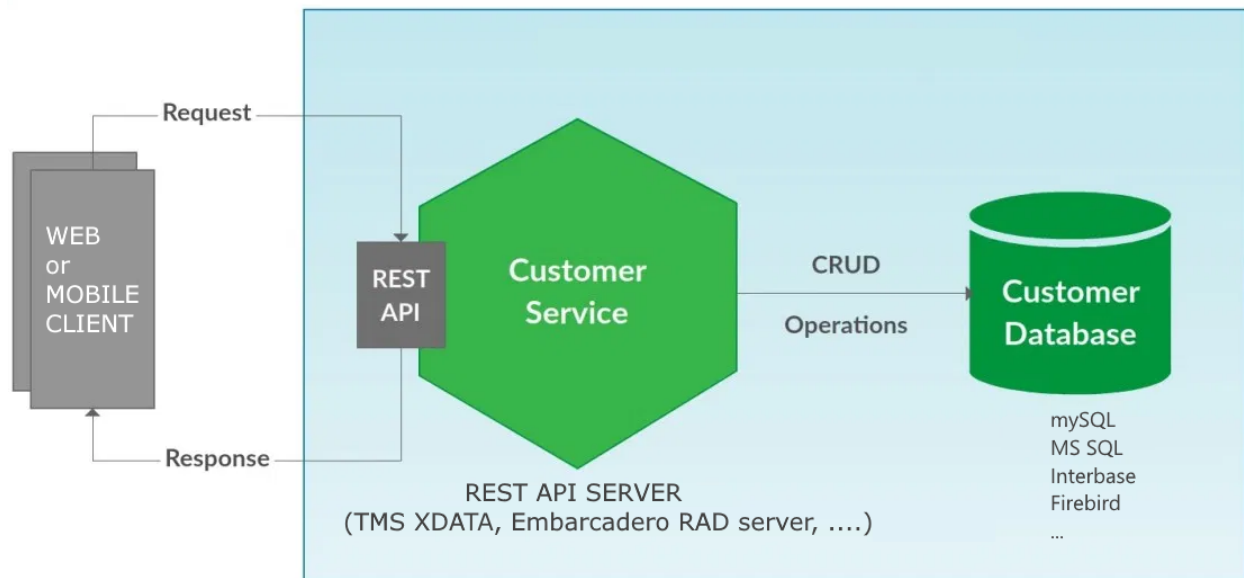
Note that you can simultaneously launch multiple live previews for multiple forms. Each browser tab will then display a preview of the form selected for live preview.

Working with databases

The TMS WEB Core framework in combination with the pas2js compiler generate web client applications that run in the browser. This means that the entire application is started from a JavaScript file and runs as such in the browser. This means that any code executed in the client eventually runs as JavaScript code in the browser. Technically, after the web server sent the JS, HTML, CSS and possibly some image resources to the browser, there is no more connection to the web server. From that moment, the web client application can start to run stand-alone in the browser. All further communication with the server is typically done via HTTP REST calls (websocket communication could be a possible alternative). For designing applications using databases, this has a number of implications:

- 1) Classic VCL database components can NOT be used, such as FireDAC, dbExpress, ADO, or other 3rd party database access components.
These are VCL components that will natively directly access the database layer on a Windows machine and the security layer of the browser would prevent such direct access anyway.
- 2) The database the web client application uses is typically not installed on the client machine. The browser shields database applications running on the client or in the network of the client machine for security reasons.
- 3) Even if a database supports a socket-based communication to perform database operations, this is typically NOT done from a web client application for security reasons. As this code is running in the browser, any experienced hacker could follow this code and could find out how to access your database and possibly invoke code himself to do malicious operations on the database.

Instead, working with databases is in this architecture of SPA's (single-page web applications) also used with Angular, Vue, React, ... done via a REST API. The web client application will authenticate & authorize against this REST API and when obtaining access, it will perform HTTP requests to perform CRUD operations on the database. It is as such the REST API server that handles the communication between client and database server and it is the REST API server that performs the database operations. The REST API server code runs on the server, can be a native application and this code cannot be seen nor affected by anyone with malicious intentions. It is very similar to a native smartphone application that connects to a central database. Also here, a typical solution is that the central database is managed by a REST API server.



TMS WEB Core is designed to be fully open with respect to the REST server providing the access to the database. As a basis, the TWebHttpRequest component can be used to perform HTTP GET/PUT/POST/DELETE requests to the REST server. Typically the REST service will expect JSON formatted data as input/output. The TJSONObject class in unit WebLib.JSON offers similar classes as offered in the System.JSON unit included standard with Delphi for VCL or FMX applications.

In addition, TMS WEB Core includes database binding mechanisms on a higher level, i.e. the level of a client dataset that will under the hood perform all necessary HTTP based communication with your REST DB server. And it includes also components for making cloud based databases accessible as datasets from your TMS WEB Core web client application.

Solutions with REST APIs for classic databases

TXDataWebDataSet	Dataset component designed for use with TMS XData REST server. This offers a code-less interface to an XData REST server with the additional advantage that XData supports meta data information. So, without additional configuration, the web client dataset TXDataWebDataSet will pickup all DB field meta data information automatically. See: https://www.tmssoftware.com/site/xdata.asp
TWebRadServerClientDataset	Dataset component designed for a REST server created with Embarcadero RAD server. Create your REST API with Embarcadero RAD server to expose CRUD operations on a

	dataset and TWebRadServerClientDataset will handle all HTTP communication and offer client-side a dataset to connect DB-aware UI controls to See: https://www.embarcadero.com/products/rad-server
TWebSQLRestClientDataset	Dataset component designed to work together with the open-source SQLDBRestBridge server. This offers a no-code configurable DB Rest Bridge server that TWebSQLRestClientDataset communicates with and offers access to via this TDataset interface. See: https://wiki.freepascal.org/SQLDBRestBridge
TWebDreamFactoryClientDataset	DreamFactory offers the creation of REST APIs, including to access databases via configuration via a web interface. So, without writing code, it is possible to create your REST API for CRUD operations on a database. When such REST API is configured in DreamFactory, the TWebDreamFactoryClientDataset can automatically communicate with it and offer database access this way via its TDataset interface to DB-aware UI controls in the TMS WEB Core web client application See: https://www.dreamfactory.com

Solutions with REST APIs for cloud database solutions

TWebMyCloudDbClientDataset	The myCloudData service offers an online cloud hosted dataset. So, here you do not need to host the database yourself, you can use an account on an already hosted database. The TWebMyCloudDbClientDataset exposes the tables on the myCloudData service as easy to use datasets. See: https://myclouddata.net
TWebFirestoreClientDataset	Google web services also includes cloud data storage with Firestore. Thanks to the TWebFirestoreClientDataset, using the Google cloud data storage becomes as seamless as possible. See https://firebase.google.com/products/firestore
TWebFaunaDbClientDataset	FaunaDB is a cloud data service company that focuses on cloud enabled data storage with data stored on the server infrastructure of FaunaDB. It offers a console for configuring the tables, queries and REST API access to it. With the TWebFaunaDbClientDataset component, using

	data hosted at FaunaDB becomes as simple as connecting DB-aware UI controls to the dataset and you are up & running.
--	--

Existing REST APIs

In many cases, there is already a REST API available to access data or services in a company that could have been created with node.js, ASP.NET core, PHP, ...

When this REST API uses OAuth2 for authentication & authorization, the component TWebRESTClient can be used for OAuth2 and performing REST HTTP requests to the server. When there is no OAUTH2 based authentication & authorization, the TWebHttpRequest component can be used for all REST HTTP GET/PUT/POST/DELETE commands.

Application

Just like in a VCL application, a TMS WEB Core application has a singleton TApplication object. The application is mainly responsible for creating and managing forms and provides in addition a couple of methods, properties and events to help in various ways. The Application object is also responsible to retrieve various formatting settings (date, time, numbers) from the browser locale.

Normally, the IDE will automatically generate the needed code so the main application form is created. Following methods, properties and events are available:

Application.CreateForm(AInstanceClass: TFormClass; var AReference);	Creates a new instance of a form class. The new form instance is returned via the AReference parameter. Note that creating a new form involves loading the form HTML file and as such, this is an asynchronous process.
Application.CreateForm(AInstanceClass: TFormClass; AElementID: string; var AReference);	Creates a new instance of a form class. The new form instance is returned via the AReference parameter. The form content is loaded in the HTML element set via ElementID. Thus, the form is hosted in the element in the form that contains it. Note that creating a new form involves loading the form HTML file and as such, this is an asynchronous process.
Application.CreateForm(AInstanceClass: TFormClass; AElementID: string; var AReference; AProc: TFormCreatedProc);	Overload of the CreateForm() method that has an extra parameter AProc. This allows to pass a procedure pointer for the procedure that will be called when the asynchronous creation of the form is ready.
Application.CreateForm(AInstanceClass: TDataModuleClass; var AReference);	Creates a new instance of a data module
Application.AppContainer: TElementID	Sets the HTML element ID for the HTML element in which TwebForm instances will be created. Default Application.AppContainer is set to 'body', putting the new created form instances in the document body.
Application.ErrorType	Defines the type of error messages that is displayed. aeSilent: non-obtrusive message in the browser console. Default when application is

	<p>compiled in release mode</p> <p>aeDialog: HTML dialog with error message centered in browser window</p> <p>aeAlert: Javascript alert with error message</p> <p>aeFooter: Rectangular area in footer of browser window containing error message</p>
Application.LoadForm(AForm: TCustomForm; AFormFile: string);	Loads the HTML file corresponding with the form instance. This is an asynchronous process.
Application.InitFormatSettings(const BrowserLocale: string);	This allows to override the automatic initialization of format settings from the default browser locale. See Appendix for possible browser locale values.
Application.Navigate(AURL: string; ATarget: TNavigationTarget);	Method to navigate from the application to a given URL. With the ATarget parameter it can be set to navigate to the URL in a new browser window or in the window where the current application is running.
Application.Download(AURL: string);	Starts the download of a file from the application from location AURL
Application.DownloadTextFile(const AText: string; AFileName: string);	Starts the download of a text file from the application with content of the text file set as AText
Application.DownloadBinaryFile(const Data: TJSUint8Array; AFileName: string);	Starts the download of a binary file from the application with content of the binary file set as Data, an array of bytes
Application.EXENAME: string	Returns the application URL
Application.GetColorScheme: TApplicationColorScheme	<p>Function returns whether the browser is running on an operating system with a regular color theme or a dark color theme.</p> <p>TApplicationColorScheme = (csNoScheme, csLight, csDark);</p> <p>csNoScheme: No operating color scheme could be detected</p> <p>csLight: Operating system has a default light color them</p> <p>csDark: Operating system has a default dark theme</p>
Application.Themed: boolean	When true, the forms in the application and its

	controls will automatically adapt their color to match the operating system default color theme being csLight or csDark.
Application.InsertCSS(CSSID, CSSFile: string);	Insert a CSS library reference with CSSID dynamically into the application main form HTML
Application.RemoveCSS(CSSID: string);	Removes the CSS library reference with CSSID dynamically from the application main form HTML
Application.ActiveForm: TForm	Returns the active form of the application
Application.AutoRouteForm: boolean	When true, a form classname can be passed as a hash on the URL and then the application will automatically create and display a form of this classname.
Application.IsOnline: boolean	Read-only property returning the online status of the application
Application.IsMobile: boolean	Read-only property returning whether the web application is running on a mobile device (smartphone or table)
Application.MainForm: TForm	Returns the main form of the application
Application.ObjectURL(Afile: TJSHtmlFile); string;	Returns a data URL from a file object
Application.RouteForm(AParameter: string);	Automatically start the form with class name passed as request parameter form=TFormClassName
Application.RunScript(AScript: string);	Executes a block of JavaScript code immediately
Application.MainForm	Returns the form instance that is the current active main form of application.
Application.Parameters: TStringList;	Returns the list of possible optional URL request parameters with which the application was started
Application.Language: TUILanguage	Sets optionally the application language. When the application language is set, it is possible that a language specific HTML file for a form is loaded. Default, Application.Language is set to INone. When Application.Language is set to a different value, the HTML file loaded for a form gets a language specific suffix. For example, when Application.Language is set to IGerman, the

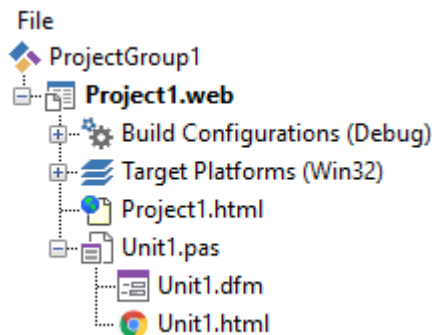
	application will load for a form in unit1.pas and having the form HTML file named unit1.html as unit1_de.html. This way, it is possible to have different language specific HTML files and have the application load the desired HTML form file when the language is set. The list of supported languages and the used language suffix is found in the appendix.
Application.ThemeColor: TColor	Gets and sets the application theme background color. This theme background color is used as form caption background color when popup forms are created.
Application.ThemeTextColor: TColor	Gets and sets the application theme text color. This theme text color is used as form caption text color when popup forms are created.
Application.ThemeButtonClassName	Gets and sets the application CSS style classname for the buttons created on dialog boxes.
Application.OnError	Event triggered when an error occurs in the application. This can be a Pascal exception or any HTML DOM specific error. The event passes the information about the error in the parameter AError: TApplicationError. When the Handled parameter is set to true, the standard error is not longer performed.
Application.OnFontCacheReady	Event triggered when fonts were successfully loaded in the cache. Font caching is used for client-side PDF generation
Application.OnHashChange	Event triggered when a hash query parameter in the application URL changes. When the application was started with myapp.html and then the URL myapp.html#newform is used, the OnHashChange event will be triggered. When used in combination with AutoRouteForm, the form classname can be passed as a hash and this form will be created when the hash changes.
Application.OnImageCacheReady	Event triggered when images that were set to be loaded by setting the URL are finished with loading asynchronously. It might be necessary to force a repaint of controls from this event.

Application.OnOnlineChange	Event triggered when the online status of the application changes. That is, when the device goes from online to offline or vice versa, this event is triggered
Application.OnOAuthToken	This event is triggered when the token is returned from a REST API service after authentication & authorization through a popup window.
Application.OnOAuthCallBack	This event is triggered when the REST API service against which authentication & authorization is performed by means of a popup window is doing a callback to the application from where it was started.

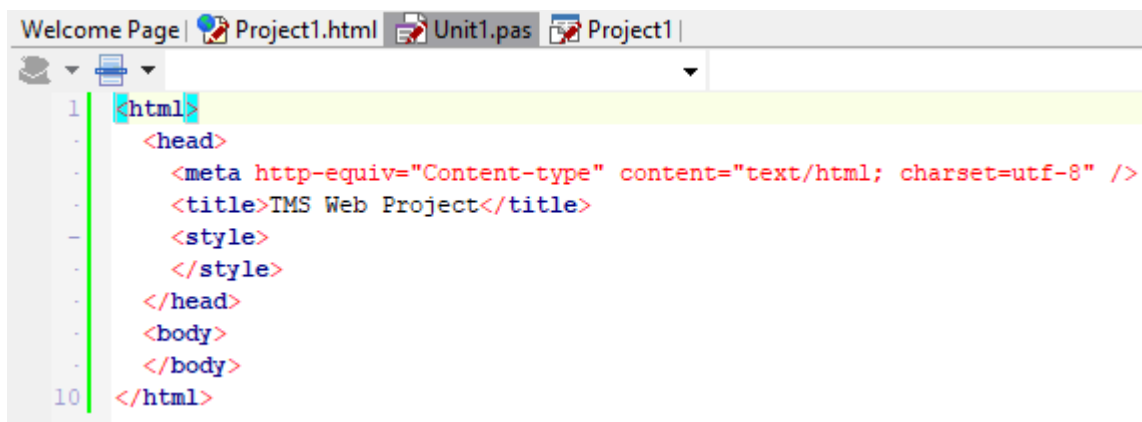
Forms

In TMS WEB Core, the base class for forms is TWebForm. TWebForm is similar to a TForm in the VCL. Controls can be put at design time on the TWebForm and will be displayed. The TWebForm is by default displayed as a full page in the browser. In addition to the controls that the form hosts that are created at design-time or at runtime, there is also the HTML code associated with the form. This HTML can be an empty HTML BODY when all controls are created by Delphi classes or it can contain additional HTML elements or HTML elements to which Delphi classes are mapped.

The default project looks like:



and you see under unit1.pas not only a reference to the DFM file where Delphi class properties are stored but also the HTML file associated with the form in Unit1. The default HTML for this form can be opened & edited from the Delphi IDE but can also be 'designed' by any other tool for creating HTML files. The default content of the HTML file is:



In the default HTML files, the BODY is empty and the controls defined in Delphi will be put in the BODY upon creation of the form instance. The application creates the main form in the same way as in a VCL application, i.e. with the code:

```
Application.CreateForm(TForm1, Form1);
```

It is also possible that the Delphi controls will be created within another HTML element than the HTML BODY element.

If unit1.html contains:

```
<html>
  <head>
    <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
    <title>TMS Web Project</title>
    <style>
    </style>
  </head>
  <body>
    Text specified directly in HTML that will be displayed in the page
    and controls will be displayed in the DIV only:
    <DIV id="form"></DIV>
  </body>
</html>
```

we can specify at form class level `Form.FormContainer: string` and set this to the HTML element ID of the HTML element in which the form should be rendered, i.e. in this case it could be set to "form".

Creating forms at runtime

Due to the asynchronous behaviour of loading the HTML for a form, the creation of a form in code is slightly different in the web than in VCL. To create a form, following code can be used:

```
procedure TForm1.WebButton1Click(Sender: TObject);
var
    newform: TForm2;

    // async called when the form is closed (via form.Close method where
    ModalResult can be set)
    procedure AfterShowModal(AValue: TModalResult);
    begin
        ShowMessage('Form 2 closed with new
value: '+newform.frm2Edit.Text);
        WebEdit1.Text := newform.frm2Edit.Text;
    end;

    // async called OnCreate for TForm2
    procedure AfterCreate(AForm: TObject);
    begin
        (AForm as TForm2).frm2Edit.Text := WebEdit1.Text;
    end;

begin
    newform := TForm2.CreateNew(@AfterCreate);
    newform.ShowModal(@AfterShowModal);
end;
```

By default, the new form TForm2 will replace the page showing the actual form. When this form is closed, the original form from where TForm2 is shown, will be displayed in the browser page again. The procedure AfterCreate is shown when the HTML for TForm2 is loaded and its controls are created. The ShowModal() method will display the actual form in the browser page and a reference to the method that will be called when the form is closed can be passed as parameter as ShowModal is not a blocking method, as such blocking methods are not possible in a browser environment.

In addition to forms displayed in the full browser page, it is also possible to create popup forms. These forms will be displayed on top of other forms, effectively disabling the forms on top of

which the new form is displayed till this new form is closed. To display a form as popup, all that is needed is setting `form.Popup = true`.

Example:

```
begin
  newform := TForm2.CreateNew(@AfterCreate);
  newform.Popup := true;
  newform.PopupOpacity := 0.2; // only needed when main form should
                               // remain visible under a layer with
                               // opacity
  newform.ShowModal(@AfterShowModal);
end;
```

For popup forms, 2 more settings are relevant.

The popup form can have a caption or not. When the popup form has a caption, the user will be able to move the popup form on the screen via the caption. The caption of the form is set via `WebForm.Caption: string`;

The popup form can be resizable (on desktop browsers) via a resizer area in the bottom-right corner of the form.

These extra form settings are done via the `TWebForm.Border` property:

WebForm.Border setting	Description
<code>fbDialogSizeable</code>	Popup form has caption and can be sized
<code>fbDialog</code>	Popup form has caption and has a fixed size
<code>fbSingle</code>	Not sizeable form, no caption

Hosting forms in other controls

Finally, it is also possible to embed other forms in controls or HTML elements in other forms. To do so, create the form with overloads of the `CreateForm` method of the `Application` object or via the `CreateNew` constructor overload of `TWebForm`:

Via the `TApplication` object:

```
procedure CreateForm(AInstanceClass: TWebFormClass; AElementID:
string; var AReference); overload;
```

```
procedure CreateForm(AInstanceClass: TWebFormClass; AElementID:
string; var AReference; AProc: TFormCreatedProc); overload;
```

The `AInstanceClass` is the class type of the form to be created. The `AElementID` is the ID of the HTML element (or Delphi class control ID) that is the HTML container in which the form will be created. The `AReference` is a reference to the form instance that will be created and optionally a reference to a procedure that will be called when the form was effectively created can be passed.

Via the `TWebForm` `CreateNew` overload:

```
constructor TWebForm.CreateNew(AElementID: string; AProc:
TFormCreatedProc);
```

The `AElementID` is the ID of the HTML element (or Delphi class control ID) that is the HTML container in which the form will be created. Optionally a method can be passed that will be called when the form was created.

Example code:

```
var
  frm: TWebForm;

procedure TfrmMain.btCreateSubFlClick(Sender: TObject);

  procedure AfterCreate(AForm: TObject);
  begin
    if Assigned(frm) and (frm is TSubForm1) then
      (frm as TSubForm1).lbTexts.Items.Assign(lbTextsMain.Items);
  end;

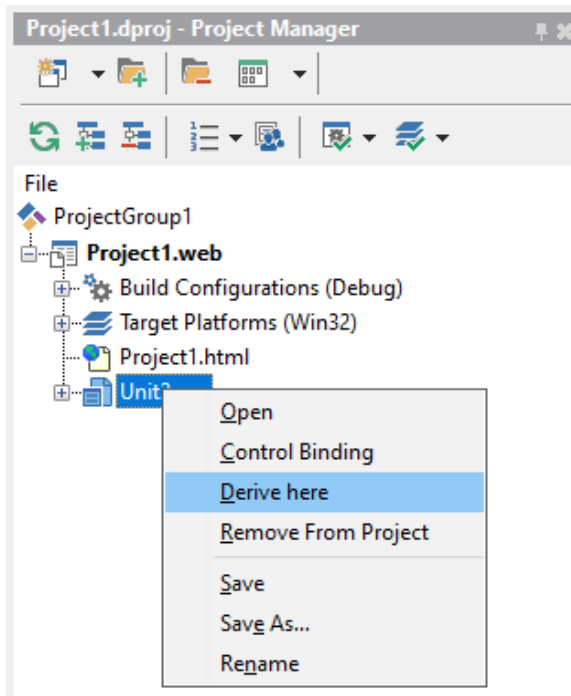
begin
  frm := TSubForm1.CreateNew(HostPanel.ElementID, @AfterCreate);
end;
```

In this sample code, a new form of the type TSubForm1 is created. The form will be displayed inside a panel on the form as we use the panel's HTML element ID. When the form is created, and this all controls on the form are accessible, the AfterCreate() procedure is called.

Form inheritance

Just like in a Delphi VCL application, TMS WEB Core web client applications can also work with the concept of visual form inheritance. This means that a TWebForm can be created, UI controls and UI control logic can be applied to this form and then a form descending from this base form class can be created. The difference with a VCL application form is that for a TWebForm, there is also an associated HTML template. As each form has a HTML template, when creating a new descending form, a new HTML template will be created. Note that for a descending form, the HTML template belonging to the descending form will be used to render the form rather than the HTML template belonging to the base form.

The process to create a descending form is done from the context menu in the Delphi IDE project manager:

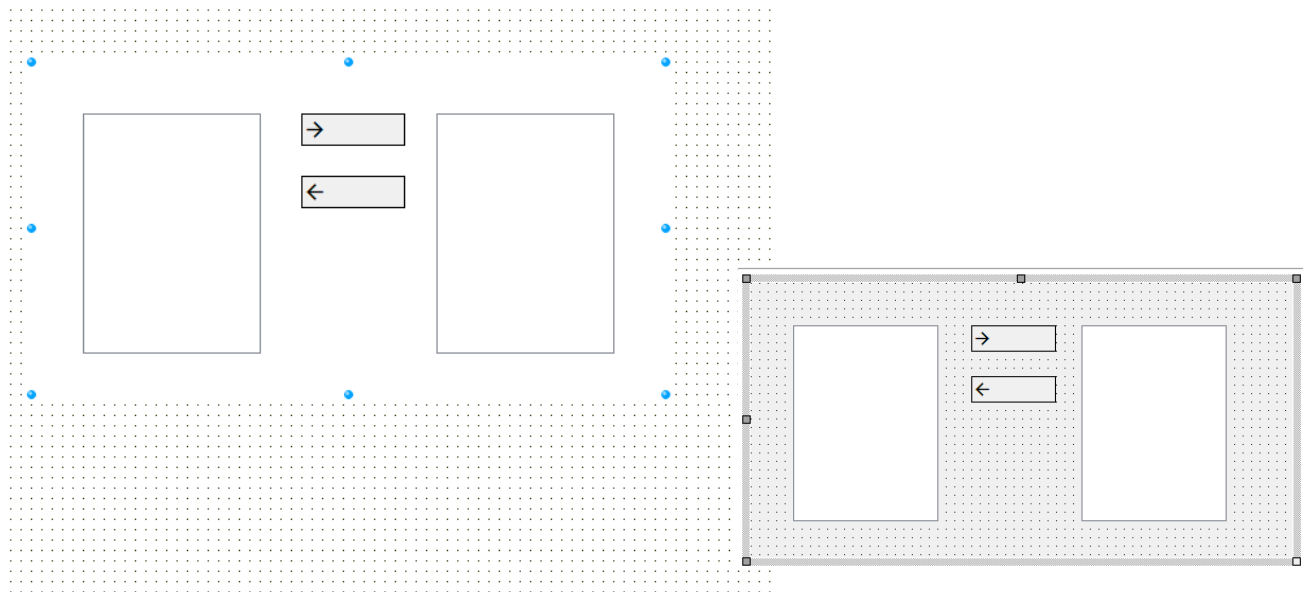


Frames

In a TMS WEB Core application, the concept for frames can also be used. Frames can encapsulate specific UI control logic in combination with UI controls. A frame in a TMS WEB Core application has no associated HTML template. The template of the form hosting the frame is used.

To create a new frame, follow the same steps as for adding a new VCL frame. After the frame is added to the project, TMS WEB Core components can be dropped on the frame and the UI control logic can be written in the frame unit.

Then, to use the frame, enter 'Frames' on the object inspector and select the frame you want to add to a form from the frames list



UI control types

TMS WEB Core supports 4 types of UI controls

UI controls encapsulating HTML elements

These are UI controls that are built-up from one or more HTML elements. All UI controls included in TMS WEB Core are of this type. In its most basic form, this is for example a `TWebButton` that maps on a HTML `<BUTTON>` element. In a more complex form, this is a `TWebLoginPanel` that consists of several `<INPUT>` elements, a `<BUTTON>` element and `<DIV>` elements.

Custom drawn controls using the HTML5 CANVAS element

These are UI controls that are based on the HTML5 CANVAS element and that are similar to VCL custom controls, custom drawn using the override of the `Paint` method. For UI interaction, the base class `TCustomControl` provides the exact same protected methods `KeyPress/KeyDown/KeyUp/MouseDown/MouseMove/MouseUp` to override. The control exposes a Canvas: `TCanvas` that has the same interface as the VCL `TCanvas`, i.e. a `Pen`, `Brush`, methods `MoveTo()`, `LineTo()`, `Rectangle()`, etc...

In addition to the VCL `TCanvas` object, it features methods to get the content of the control as image or to download it as image:

```
TCanvas.GetBase64Image: string;  
TCanvas.DownloadImage( AFileName: string; AType: TImageType = itPNG);  
function GetAsImage(AType: TImageType): string;
```

With `TImageType = (itBase64, itBMP, itPNG, itJPEG, itGIF);`

TMS FNC controls

The TMS FNC component framework is an abstraction layer that facilitates writing UI controls with a single code base that can be used for VCL, FMX, LCL and also TMS WEB Core applications. Several TMS FNC products, i.e. TMS FNC Chart, TMS FNC UI Pack and TMS FNC Dashboard Controls Pack support to use of the components also in web applications. For documentation about FNC controls, this is included in the different TMS FNC products and

all documentation that applies to use of the controls in VCL, FMX or LCL applications also applies to use of the controls in TMS WEB Core applications.

jQuery UI controls

Several controls are provided that are actually Pascal wrapper classes for underlying jQuery UI controls. This includes a set of Pascal wrapper classes for the jqWidget controls (www.jqwidgets.com)

Standard Components

TMS WEB Core comes with a lot of components out of the box enabling you to go ahead immediately creating web applications from the Delphi IDE. Many of these standard controls resemble VCL controls and great care has been taken to give these controls class names and properties, methods and event handlers that match their VCL counterparts. This has been done to make the learning curve for Delphi developers used to create Windows VCL applications as light as possible to create web applications. The standard controls have the prefix “TWeb”, i.e. where in VCL a TButton is used, there is in TMS WEB Core a component TWebButton. Where there is in VCL a TListBox, in TMS WEB Core, its counterpart is TWebListBox etc...

Common properties of visual controls

Visual controls are descending from TControl. For controls without a Canvas, i.e. controls that map directly on a hierarchy of HTML elements (excluding the HTML5 CANVAS element), TWinControl descending from TControl is defined. Controls doing custom painting are descending from TCustomControl that descends from TControl. Finally, when the control does not need user-interface interaction via mouse or keyboard, the TGraphicControl is introduced that descends from TCustomControl. At TControl level, a number of properties is introduced that are then further common for all descending user interface controls.

Align	Sets the alignment of the control in relationship to its parent control: alLeft: control aligns to the left-side of its parent alTop:control aligns to the top of its parent alBottom:aligns to the bottom of its parent alRight:aligns to the right-side of its parent alClient:aligns to the client-size of its parent
AlignWithMargin	When true, the margins settings are taking in account for calculating the alignment
Anchors	Gets or sets the anchoring of the control. Values can be akLeft akTop akRight akBottom
Cursor	Sets the mouse cursor used when the mouse

	is over the control
Enabled	Sets the control to enabled or disabled
ElementClassName	Sets the CSS class name(s) for the HTML element used to represent the control
ElementFont	Sets whether the control Font property is used to set the font (efProperty) or CSS will control the font (efCSS)
ElementID	Sets the HTML ID of the HTML element already present in the HTML document that the Pascal class needs to connect to (instead of creating a new HTML element instance)
ElementPosition	Sets the position of the element in the HTML page as epAbsolute, epRelative or epNone.
EventStopPropagation: TEventPropagation	<p>Different from the Windows operating system, a HTML element event such as for mouse and keyboard are sent to the element but also to the parent element and so on by default. This is for UI controls typically not desired and as such for TMS WEB Core controls by default disabled. However, with the EventStopPropagation, you can control what event are propagated and what not. EventStopPropagation: TEventPropagation is defined as:</p> <pre>TElementEvent = (eeClick, eeMouseDown, eeMouseUp, eeMouseMove, eeDbClick, eeKeyPress, eeKeyDown, eeKeyUp); TEventPropagation = set of TElementEvent;</pre> <p>As such, by default it is initialized to:</p> <pre>EventStopPropagation := [eeClick, eeDbClick, eeMouseUp, eeMouseMove, eeMouseDown, eeKeyPress, eeKeyDown, eeKeyUp];</pre> <p>so all event propagation is blocked.</p> <p>To allow all event propagation, you would set</p> <pre>Control.EventStopPropagation := [];</pre>
Height	Absolute height value for the control
HeightPercent	Height value used when HeightStyle is

	ssPercent
HeightStyle	When HeightStyle is set to ssAbsolute, the Height value is used to set the absolute height of the container HTML element of the control. When HeightStyle is set to ssPercent, the HeightPercent value is used. When HeightStyle is set to ssNone, no height is specified on the container HTML element, meaning it will auto size.
Hint	Sets the hint value for the container HTML element
Margins	Sets the margin values
ShowHint	When true, the hint is used for the control
Visible	When true, the control is visible
Width	Absolute width value for the control
WidthPercent	Width value used when WidthStyle is ssPercent
WidthStyle	When WidthStyle is set to ssAbsolute, the Width value is used to set the absolute width of the container HTML element of the control. When WidthStyle is set to ssPercent, the WidthPercent value is used. When WidthStyle is set to ssNone, no width is specified on the container HTML element, meaning it will auto size.

Common events of visual controls

OnClick	Event triggered on a mouse click on the control
OnDbClick	Event triggered on a mouse double click on the control
OnEnter	Event triggered when control gets focus
OnExit	Event triggered when focus leaves control
OnMouseDown	Event triggered when mouse goes down on control
OnMouseMove	Event triggered when mouse moves over control
OnMouseUp	Event triggered when mouse goes up on

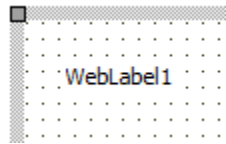
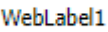
	control
OnMouseEnter	Event triggered when mouse enters control
OnMouseLeave	Event triggered when mouse leaves control
OnDragDrop	Event triggered when a drop happens on control during drag operation
OnDragOver	Event triggered when a mouse drag is occurring over the control
OnStartDrag	Event triggered when a drag operation starts
OnEndDrag	Event triggered when a drag operation ends

TWebLabel



Description

Below is a list of the most important properties methods and events for the TWebLabel.
TWebLabel is a label control similar to a VCL TLabel.

 <p>Designtime</p>	 <p>Runtime</p>
---	--

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebLabel

Alignment	Sets the alignment of the text within the label control
AutoSize	When true, the size of the label adapts to the text in the label
Caption	Sets the text for the label
ElementClassName	Optionally sets the CSS classname for the outer DIV element of the label when styling via CSS is used
ElementLabelClassName	Optionally sets the CSS classname for the label HTML element
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
EllipsisPosition	<p>Sets the type of ellipsis to use for showing the text when it doesn't fit in the label rectangle.</p> <p>epNone: no ellipsis used</p> <p>epEndEllipsis: ellipsis at the end of the text</p> <p>epPathEllipsis: label text contains a path name and ellipsis is set taking a file path in account</p> <p>epWordEllipsis: ellipsis is positioned at word boundary</p>
HTMLType	<p>Sets the type of the HTML element that is created for the label. This can be:</p> <p>tDIV</p> <p>tH1..tH6</p> <p>tLABEL</p> <p>tSPAN</p> <p>The respective HTML elements created within the label will be</p> <p><DIV><H1>..<<H6><LABEL></p>
Layout	<p>Sets the vertical text position in the label</p> <p>tlTop: top aligned</p> <p>tlCenter: center aligned</p>

	tlBottom: bottom aligned
Transparent	When true, no background color is used
WordWrap	When true, the text is rendered with automatic wordwrap

Events for TWebLabel

OnClick	Triggered when the mouse is clicked on the label
OnDbClick	Triggered when the mouse is double-clicked on the label

TWebButton



Description

Below is a list of the most important properties methods and events for TWebButton. TWebButton is a button control similar to a VCL TButton control.

 <p>Design-time</p>	 <p>Runtime</p>
--	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the id attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<BUTTON ID="UniqueID"></BUTTON>
ElementID	UniqueID

Properties for TWebButton

ButtonType	Allows to set the type attribute for the HTML button element when needed (button submit reset)
Caption	Sets the text for the button
ElementClassName	Optionally sets the CSS classname for the button when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML BUTTON element in the form HTML file the button needs to be connected with. When connected, no new button is created but the Delphi class is connected with the existing HTML element in the form HTML file
TabOrder	Sets the tab order of the control
TabStop	When true, the focus is turned to the control when pressing tab
TextDirection	Sets the text direction to tdDefault: does not use direction attribute tdInherit: uses TextDirection of parent control tdRightToLeft:uses rtl direction attribute tdLeftToRight: uses ltr direction attribute

Events for TWebButton

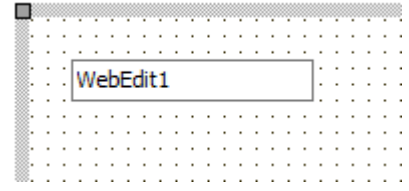

OnClick	Event is triggered when the button is clicked
OnEnter	Event triggered when the button gets focus
OnExit	Event triggered when the focus leaves the button

TWebEdit



Description

Below is a list of the most important properties methods and events for TWebEdit. TWebEdit is an edit input control similar to a TEdit in VCL.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebEdit

AutoCompletion	<p>Sets the auto completion type that the browser uses to fill the edit controls on a form based on its cache of entries previously made. Note that in order to have autocompletion working, it is required that the TWebEdit control is placed on a TWebHTMLForm.</p> <p>Set value to acNone to force no autocompletion suggestion from the browser.</p>
----------------	---

AutoFocus	When true and the control is the first control in the tab order, it will display focused
BorderStyle	Sets the border style for the control
EditType	Sets the allowed type of characters that can be entered. Options are weFloat, weHex, weNumeric, weSignedFloat, weSignedNumeric, weString, weSearch
ElementClassName	Optionally sets the CSS classname for the edit control when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML INPUT element in the form HTML file the edit control needs to be connected with. When connected, no new edit control is created but the Delphi class is connected with the existing HTML element in the form HTML file
Pattern	Sets the pattern of accepted characters for form validation (used when TWebEdit is used on a TWebHTMLForm)
Required	When true, the content of the TWebEdit being empty will cause a validation popup when used on a TWebHTMLForm.
SelLength	Gets or sets the length of selected text in the edit control
SelStart	Gets or sets the caret position in the edit control
ShowFocus	When true, the border color changes when the control has focus
TabOrder	Sets the tab order of the control
TabStop	When true, the focus is turned to the control when pressing tab
Text	Gets or sets the text of the edit control
TextDirection	Sets the text direction to tdDefault: does not use direction attribute tdInherit: uses TextDirection of parent control tdRightToLeft: uses rtl direction attribute tdLeftToRight: uses ltr direction attribute

Methods for TWebEdit

Clear	Removes text from the edit
-------	----------------------------

ClearSelection	Removes the selected text from the edit
----------------	---

Events for TWebEdit

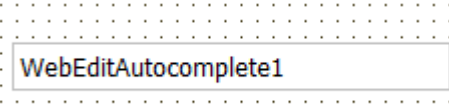
OnClick	Event triggered when the control is clicked
OnChange	Event triggered when the value in the edit control is changed via the UI
OnEnter	Event triggered when the control gets focus
OnExit	Event triggered when focus leaves the control

TWebEditAutocomplete



Description

Below is a list of the most important properties methods and events for TWebEditAutocomplete. TWebEditAutocomplete is an edit control with the possibility to display a list of predefined values in a popup, filtered based on user input.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

Note that for the popup to be displayed at the correct position, it is required to add a set the CSS position property to relative or absolute.

HTML tag	<DIV ID="UniqueID" style="position:relative"></DIV>
ElementID	UniqueID

Properties for TWebEditAutocomplete

ActiveItemClassName	Optionally sets the CSS classname for the active item in the popup list when styling via CSS is used
BorderStyle	Sets the border style for the control
ElementClassName	Optionally sets the CSS classname for the edit control when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML INPUT element in the form HTML file the edit control needs to be connected with. When connected, no new edit control is created but the Delphi class is connected with the existing HTML element in the form HTML file
Items	The list of pre-defined values to display in the popup
ItemIndex	Gets the currently selected item index
ItemClassName	Optionally sets the CSS classname for the items in the popup list when styling via CSS is used
LookupCaseSensitive	Sets if the lookup search is case sensitive
LookupMinLength	The minimum text length required before the lookup is initiated.
LookupType	The type of lookup search that is performed. ItAnywhere: Search for the character(s) anywhere in the text

	ItFirstCharacter: Search for the character(s) in the beginning of the text
PopupClassName	Optionally sets the CSS classname for the the popup styling via CSS is used
PopupHeight	Sets the height of the popup. If the number of visible items exceeds the height of the popup a scrollbar is displayed. Set to 0 to let the popup autosize based on the number of displayed items.
TabOrder	Sets the tab order of the control
TabStop	When true, the focus is turned to the control when pressing tab
Text	Gets or sets the text of the edit control

Events for TWebEditAutocomplete

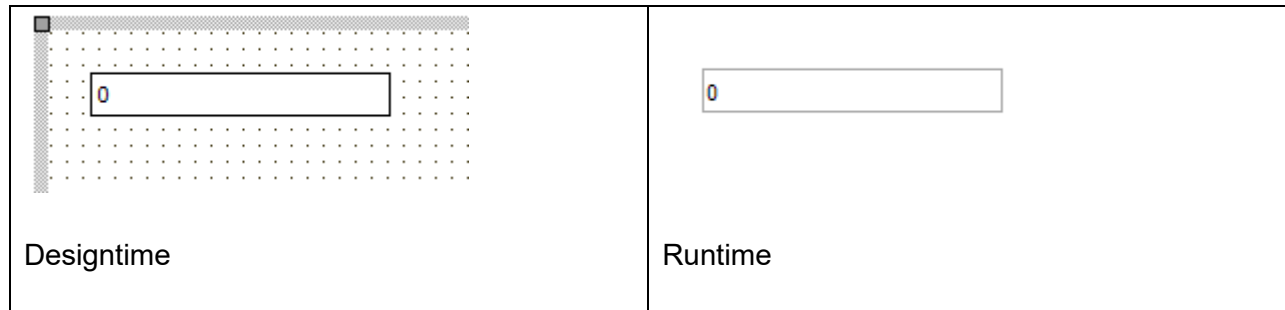
OnClick	Event triggered when the control is clicked
OnChange	Event triggered when the value in the edit control is changed via the UI
OnEnter	Event triggered when the control gets focus
OnExit	Event triggered when focus leaves the control
OnRenderItem	Event triggered for each displayed item. Items can be customized with the Args.ItemElement parameter
OnSelect	Event triggered when an item is selected from the popup list

TWebSpinEdit



Description

Below is a list of the most important properties methods and events for TWebSpinEdit.
TWebSpinEdit is an edit control with an embedded spin up & down button, similar to a VCL TSpinEdit. The TWebSpinEdit requires a fully HTML5 compliant browser.



HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<INPUT TYPE="NUMBER" ID="UniqueID">
ElementID	UniqueID

Properties for TWebSpinEdit

AutoSize	When true, the width of the control adapts to the text
BorderStyle	Sets the border style for the control
ElementClassName	Optionally sets the CSS classname for the spin control when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the spin control needs to be connected with. When connected, no new spin contrl is created but the Delphi class is connected with the existing HTML element in the form HTML file
Increment	Gets or sets the step to increment the value with the up/down buttons
ShowFocus	When true, the border color changes when the control has focus

TabOrder	Sets the tab order of the control
TabStop	When true, the focus is turned to the control when pressing tab
TextDirection	Sets the text direction to tdDefault: does not use direction attribute tdInherit: uses TextDirection of parent control tdRightToLeft: uses rtl direction attribute tdLeftToRight: uses ltr direction attribute
Value	Sets or gets the value of the control

Events for TWebSpinEdit

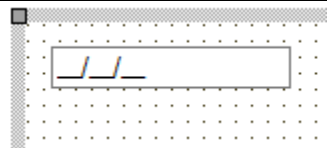

OnChange	Event triggered when the value of the spin edit control changes from the UI
OnClick	Event triggered when the control is clicked
OnDbClick	Event triggered when the control is double-clicked
OnEnter	Event triggered when the control gets focus
OnExit	Event triggered when focus leaves the control

TWebMaskEdit



Description

Below is a list of the most important properties methods and events for TWebMaskEdit. TWebMaskEdit is an edit control with a edit mask capability that controls what character(s) can be typed at what position in the edit control, similar to a VCL TMaskEdit.

 <p>EditMask property set to: !99/99/00;1;_</p>	 <p>Runtime</p>
--	---

DesignTime	
------------	--

Properties for TWebMaskEdit

Alignment	Sets the alignment of the entered text in the edit control
AutoSelect	When true, all text gets selected when the control gets focus
AutoSize	When true, the width of the control adapts to the text
BorderStyle	Sets the border style for the control
CharCase	Controls whether characters are automatically entered as lowercase, uppercase or entered as typed.
EditMask	Sets the mask for the inplace editor. The mask that can be used for the TWebMaskEdit is compatible with the mask available for a VCL TMaskEdit control. The description of the mask capabilities can be found here: http://docwiki.embarcadero.com/Libraries/Tokyo/en/System.MaskUtils.TEditMask
EditText	Sets & gets the value of the edit control without taking the mask in account
ElementClassName	Optionally sets the CSS classname for the spin control when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the spin control needs to be connected with. When connected, no new spin contrl is created but the Delphi class is connected with the existing HTML element in the form HTML file
ShowFocus	When true, the border color changes when the control has focus
TabOrder	Sets the tab order of the control
TabStop	When true, the focus is turned to the control when pressing tab
Text	Sets or gets the text in the edit control

Events for TWebMaskEdit

OnChange	Event triggered when the value of the spin edit control changes from the UI
OnClick	Event triggered when the control is clicked
OnDbClick	Event triggered when the control is double-clicked
OnEnter	Event triggered when the control gets focus
OnExit	Event triggered when focus leaves the control
OnKeyDown	Event triggered on key down in the edit

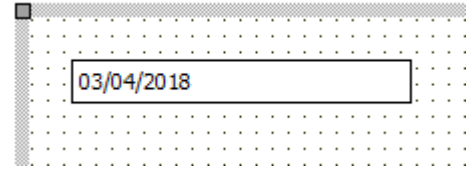

	control
OnKeyPress	Event triggered on key press in the edit control
OnKeyUp	Event triggered on key up in the edit control

TWebDateTimePicker



Description

Below is a list of the most important properties methods and events for the TWebDateTimePicker. TWebDateTimePicker allows to select a date or time, similar to a date/time picker in VCL. This control requires a fully HTML5 compliant browser.

 <p>Design-time</p>	 <p>Runtime</p>
--	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<INPUT TYPE="DATE" ID="UniqueID">
ElementID	UniqueID

Properties for TWebDateTimePicker

BorderStyle	Sets the border style for the control
Checked: boolean	Sets or gets the checkbox state of the checkbox in the datepicker when ShowCheckBox = true
Date	Gets or sets the date value of the control
ElementClassName	Optionally sets the CSS classname for the date/time picker when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the date picker needs to be connected with. When connected, no new date picker is created but the Delphi class is connected with the existing HTML element in the form HTML file
Kind	Configures the control as date or as time picker
ShowCheckBox: boolean	When true, a checkbox is shown in front of the date/time picker to enable/disable it
ShowFocus	When true, the border color changes when the control has focus
TabOrder	Sets the tab order of the control
TabStop	When true, the focus is turned to the control when pressing tab
TextDirection	Sets the text direction to tdDefault: does not use direction attribute tdInherit: uses TextDirection of parent control tdRightToLeft: uses rtl direction attribute tdLeftToRight: uses ltr direction attribute
Time	Gets or sets the

Events for TWebDateTimePicker

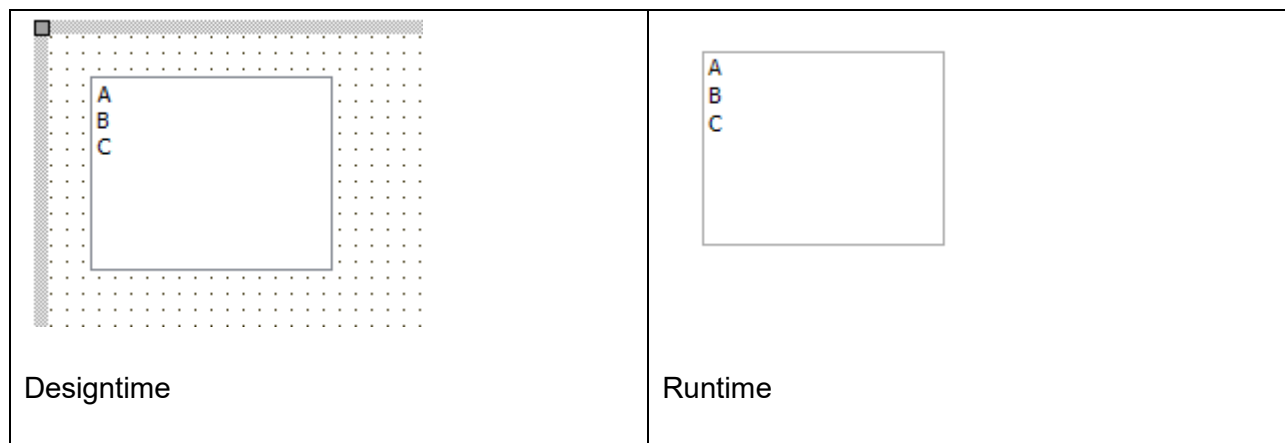
OnChange	Event triggered when the date or time changes via the UI
OnClick	Event triggered when the control is clicked
OnDbClick	Event triggered when the control is double-clicked
OnEnter	Event triggered when the control gets focus
OnExit	Event triggered when focus leaves the control

TWebListBox



Description

Below is a list of the most important properties methods and events for TWebListBox. A TWebListBox is a control having a list of (text) items, similar to a VCL TListBox.



HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<SELECT ID="UniqueID"></SELECT>
ElementID	UniqueID

Properties for TWebListBox

BorderStyle	Sets the border style for the control
ElementClassName	Optionally sets the CSS classname for the

	date/time picker when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the date picker needs to be connected with. When connected, no new date picker is created but the Delphi class is connected with the existing HTML element in the form HTML file
Enabled	Sets whether the control is enabled or disabled
ItemHeight	Sets the height of individual items in the listbox
ItemIndex	Sets or gets the index of the selected item
Items	Access to the items in the listbox as a TStringList
MultiSelect	When true, multiple items can be selected in the listbox
ShowFocus	When true, the border color changes when the control has focus
TabOrder	Sets the tab order of the control
TabStop	When true, the focus is turned to the control when pressing tab
TextDirection	Sets the text direction to tdDefault: does not use direction attribute tdInherit: uses TextDirection of parent control tdRightToLeft: uses rtl direction attribute tdLeftToRight: uses ltr direction attribute

Events for TWebListBox

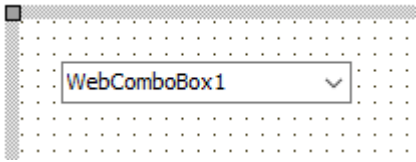

OnChange	Event triggered when the selected item changes in the listbox
OnClick	Event triggered when the listbox is clicked
OnDbClick	Event triggered when the listbox is double-clicked
OnEnter	Event triggered when the control gets focus
OnExit	Event triggered when focus leaves the control

TWebComboBox



Description

Below is a list of the most important properties methods and events for TWebComboBox. A TWebComboBox is a control having a list of (text) items, similar to a VCL TComboBox

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

For a combobox with Style = csDropDown

HTML tag	<INPUT ID="UniqueID"></SELECT>
ElementID	UniqueID

For a combobox with Style = csDropDownList

HTML tag	<SELECT ID="UniqueID"></SELECT>
ElementID	UniqueID

Properties for TWebComboBox

ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the

	label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
ItemIndex	Sets or gets the index of the selected item
Items	Access to the items in the listbox as a TStringList
ShowFocus	When true, the border color changes when the control has focus
TabOrder	Sets the tab order of the control
TabStop	When true, the focus is turned to the control when pressing tab
Text	Gets or sets the selected value in the combobox
TextDirection	Sets the text direction to tdDefault: does not use direction attribute tdInherit: uses TextDirection of parent control tdRightToLeft: uses rtl direction attribute tdLeftToRight: uses ltr direction attribute

Events for TWebComboBox

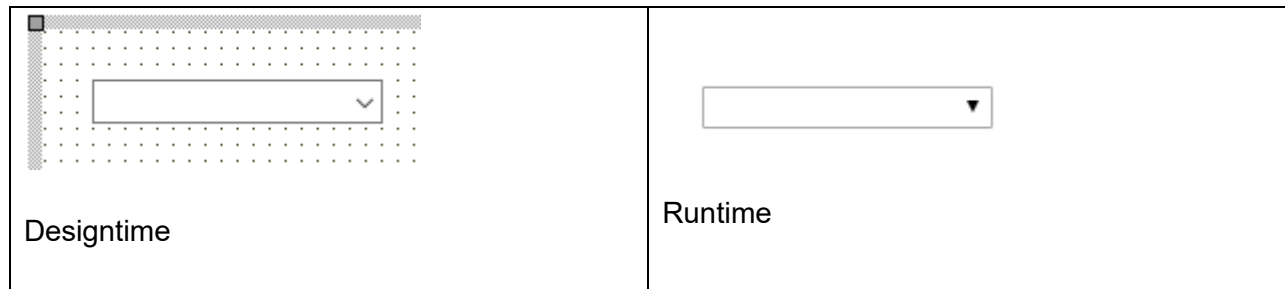
OnChange	Event triggered when the selected item changes in the listbox
OnClick	Event triggered when the listbox is clicked
OnDbClick	Event triggered when the listbox is double-clicked
OnEnter	Event triggered when the control gets focus
OnExit	Event triggered when focus leaves the control

TWebLookupComboBox



Description

Below is a list of the most important properties methods and events for TWebLookupComboBox. A TWebLookupComboBox is a control having a list of (text) items, similar to a VCL TComboBox



HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<SELECT ID="UniqueID"></SELECT>
ElementID	UniqueID

Properties for TWebLookupComboBox

DisplayText: string	Public property returning the selected item DisplayText
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
ItemIndex	Sets or gets the index of the selected item
LookupValues	Access to a collection of items of the TWebLookupComboBox where each item has DisplayText: string and Value: string property.

ShowFocus	When true, the border color changes when the control has focus
TabOrder	Sets the tab order of the control
TabStop	When true, the focus is turned to the control when pressing tab
Text	Gets or sets the selected value in the combobox
TextDirection	Sets the text direction to tdDefault: does not use direction attribute tdInherit: uses TextDirection of parent control tdRightToLeft: uses rtl direction attribute tdLeftToRight: uses ltr direction attribute
Value: string	Public property returning the selected item value.

Events for TWebLookupComboBox

OnChange	Event triggered when the selected item changes in the listbox
OnClick	Event triggered when the listbox is clicked
OnDbClick	Event triggered when the listbox is double-clicked
OnEnter	Event triggered when the control gets focus
OnExit	Event triggered when focus leaves the control

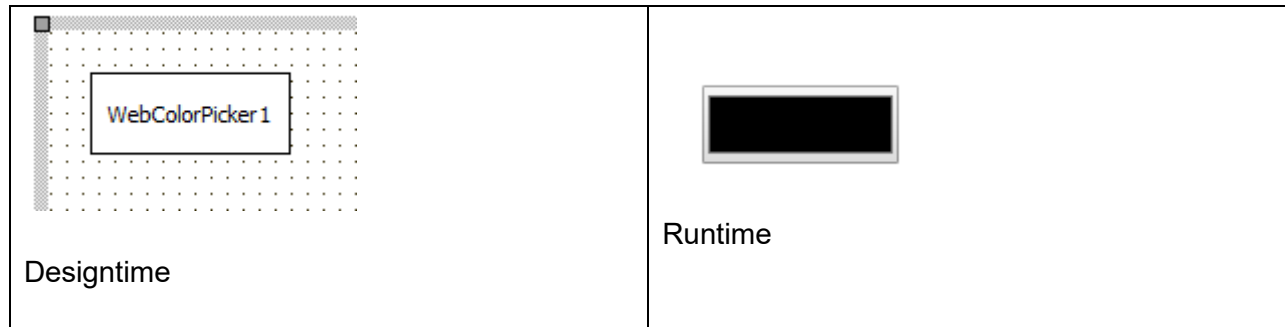
Add DisplayText/Value pairs to the TWebLookupCombobox with
TWebLookupCombobox.LookupValues.AddPair(AValue, ADisplayText);

TWebColorPicker



Description

Below is a list of the most important properties methods and events for TWebColorPicker. TWebColorPicker is a control to allow selecting a color. Note that a browser with full HTML5 compliance is needed for this control.



HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<INPUT TYPE="COLOR" ID="UniqueID">
ElementID	UniqueID

Properties for TWebColorPicker

Color	Gets or sets the selected color of the color picker
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file

Events for TWebColorPicker

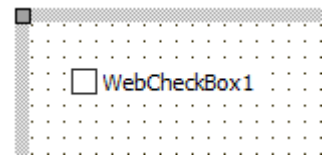
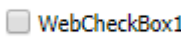
OnSelect	Event triggered when a color is selected via the color picker
----------	---

TWebCheckBox



Description

Below is a list of the most important properties methods and events for TWebCheckBox, TWebCheckBox represents a two-state checkbox or three-state checkbox and is similar to a VCL TCheckBox

 <p>Designtime</p>	 <p>Runtime</p>
---	---

Properties for TWebCheckBox

Caption	Sets or gets the text for the checkbox
Checked	Sets or gets the checkbox state
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
ShowFocus	When true, the border color changes when the control has focus
State	Allows to get or set the checkbox state in three states: checked, unchecked, grayed.
TabOrder	Sets the tab order of the control
TabStop	When true, the focus is turned to the control when pressing tab

Events for TWebCheckBox

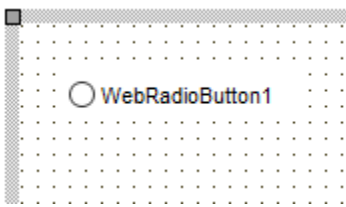
OnClick	Event triggered when the checkbox is clicked
---------	--

TWebRadioButton



Description

Below is a list of the most important properties methods and events for TWebRadioButton, TWebRadioButton represents a two-state checkbox or three-state checkbox and is similar to a VCL TRadioButton

 <p>Designtime</p>	 <p>Runtime</p>
--	---

Properties for TWebRadioButton

Caption	Sets or gets the text for the radiobutton
Checked	Sets or gets the radiobutton state
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
ShowFocus	When true, the border color changes when

	the control has focus
TabOrder	Sets the tab order of the control
TabStop	When true, the focus is turned to the control when pressing tab

Events for TWebRadioButton

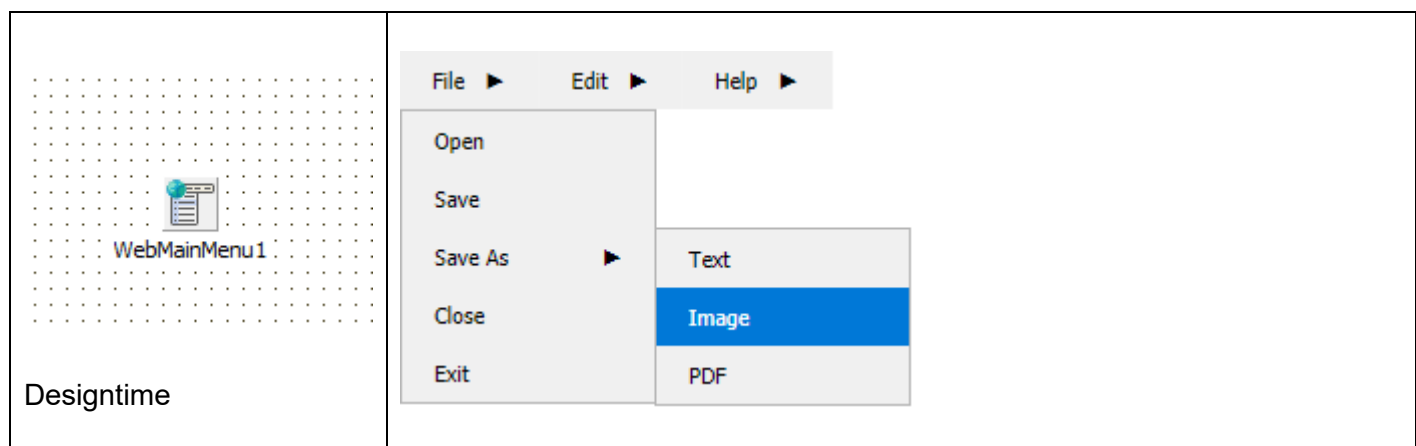
OnClick	Event triggered when the radiobutton is clicked
---------	---

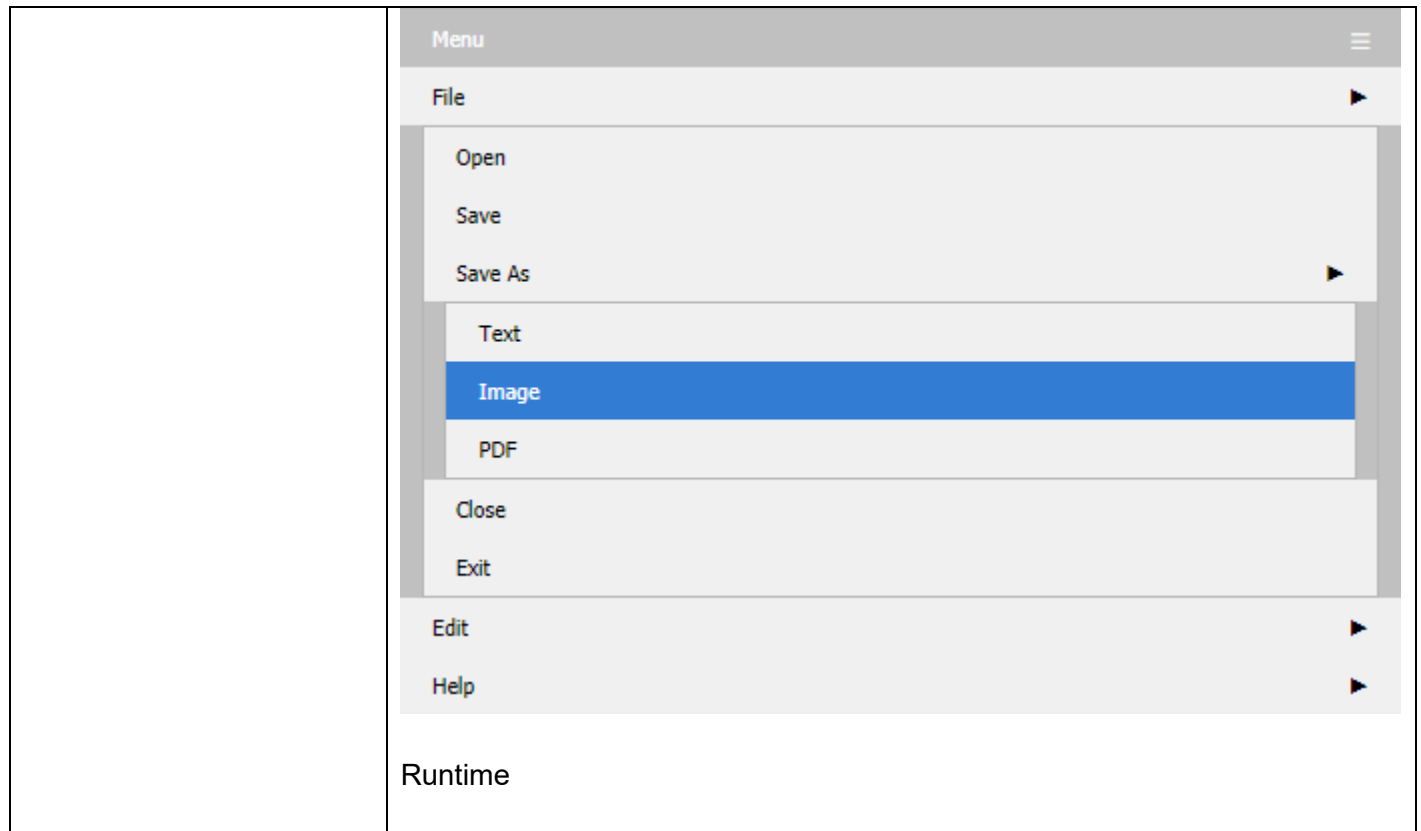
TWebMainMenu



Description

Below is a list of the most important properties methods and events for TWebMainMenu. Represents a menu control with support for sub-menus. The TWebMainMenu optionally can be displayed as a vertical hamburger menu. By default the menu automatically transforms into a hamburger menu if the available browser width is 768 pixels or less. This behavior can be customized with the HamburgerMenu properties.





HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebMainMenu

Appearance	
BackgroundColor	Sets the background color of the menu
HamburgerMenu	
BackgroundColor	Sets the background color the menu

	caption
Caption	Sets the caption text
CaptionColor	Sets the caption font color
ResponsiveMaxWidth	Sets the maximum browser window width for which the hamburger menu is displayed
Visible	<p>Sets when the hamburger menu is visible.</p> <p>hmAlways: The menu is always displayed as a hamburger menu regardless of window width</p> <p>hmNever: the hamburger menu is never displayed and the default main menu is always displayed.</p> <p>hmResponsive: the hamburger menu is only displayed when the available browser window width is 768 pixels or less.</p>
HoverColor	Sets the background color of a hovered menu item
HoverFontColor	Sets the font color of a hovered menu item
ImageSize	Sets the size of the image if available
ImageURLs	Set the list of images available to use for menu items. Set the MenuItem.ImageIndex value to the index of the image that should be displayed in the menu item.
SubmenuIndicator	Sets the symbol used to indicate a submenu is available
Container	Sets the external control the menu is displayed in. For example, a TWebPanel control. By default, the menu is displayed in the top left corner of the browser window.
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
Items	
Caption	The caption text
Checked	Sets if a checkmark is displayed next to the caption text

Enabled	Sets if the item is enabled
ImageIndex	Sets the index of the image from the Appearance.imageURLs list that should be displayed next to the caption text
RadiolItem	Sets if the menu item should be displayed as a radiobutton
Visible	Sets if the menu item is visible

Events for TWebMainMenu

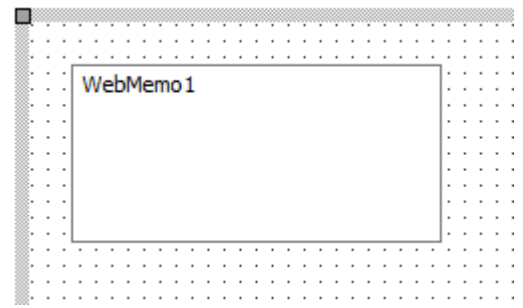
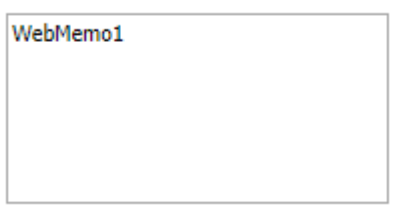
OnChange	Event triggered when a menu item is clicked.
----------	--

TWebMemo



Description

Below is a list of the most important properties methods and events for the TWebMemo. TWebMemo is a multiline editable control, similar to a VCL TMemo. It is based on the HTML TEXTAREA element.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<TEXTAREA ID="UniqueID"></TEXTAREA>
ElementID	UniqueID

Properties for TWebMemo

AutoSize	When true, the size of the control will automatically adapt to the text in the memo
BorderStyle	Sets the border style of the control
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
Lines	Access to the content of the memo via a TStringList property
SelLength	Gets or sets the length of the selection in the memo
SelStart	Gets or sets the selection starting point in the memo
ShowFocus	When true, the border color changes when the control has focus
TabOrder	Sets the tab order of the control
TabStop	When true, the focus is turned to the control when pressing tab
TextDirection	Sets the text direction to tdDefault: does not use direction attribute tdInherit: uses TextDirection of parent control tdRightToLeft:uses rtl direction attribute tdLeftToRight: uses ltr direction attribute

Methods for TWebMemo

Clear	Removes text from the memo
ClearSelection	Removes the selected text from the memo

Events for TWebMemo

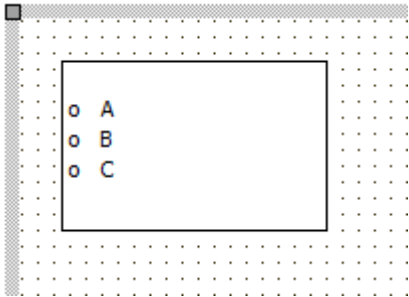
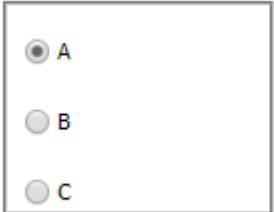
OnChange	Event triggered when the content of the memo changes
OnClick	Event triggered when the memo is clicked
OnDbClick	Event triggered when the memo is double-clicked

TWebRadioGroup



Description

Below is a list of the most important properties methods and events for the TWebRadioGroup. TWebRadioGroup is a group of radio button controls similar to a VCL TRadioGroup.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

Properties for TWebRadioGroup

Caption	Sets the caption text of the radiogroup
Columns	Defines in how many columns the radiobuttons are displayed. Default is 1.
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
ItemIndex	Sets or gets the selected radio button in the group
Items	Access to the radio button captions in the group via a TStringList property

Events for TWebRadioGroup

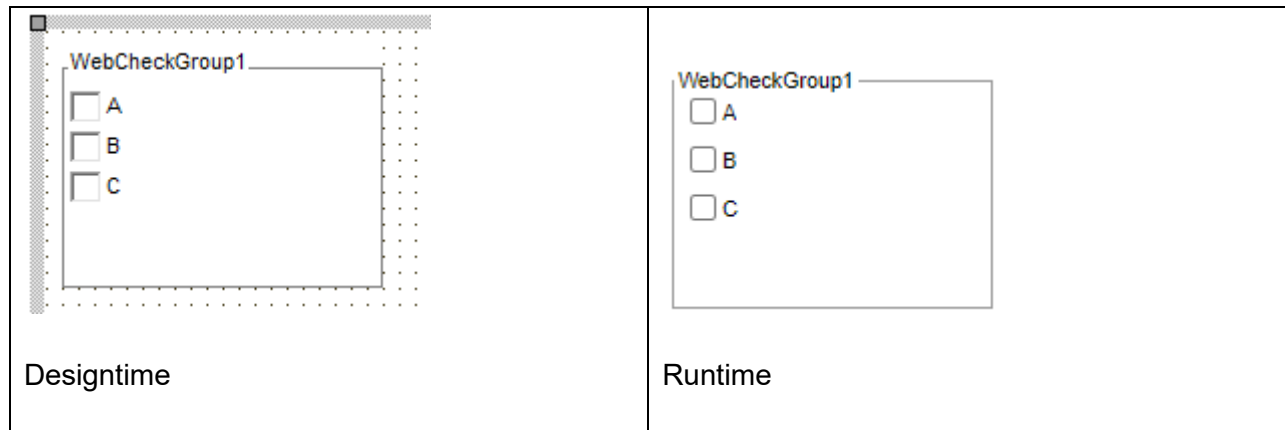
OnChange	Event triggered when the selected radio button in the radiogroup changes
----------	--

TWebCheckGroup



Description

Below is a list of the most important properties methods and events for the TWebCheckGroup. TWebCheckGroup is a group of checkbox controls similar to a VCL TCheckGroup.



Properties for TWebCheckGroup

Caption	Sets the caption text of the checkgroup
Checked[AIndex: integer]: boolean	Gets or sets the checkbox state of a checkbox in the group with index AIndex
Columns	Defines in how many columns the checkboxes are displayed. Default is 1.
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
Items	Access to the checkbox captions in the group via a TStringList property

Events for TWebCheckGroup

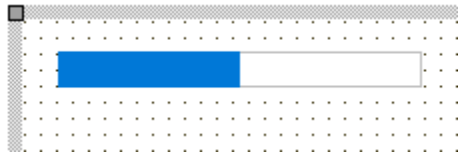

OnCheckClick	Event triggered when a checkbox is toggled. It returns the index of the checkbox
--------------	--

TWebProgressBar



Description

The TWebProgressBar is a progress indicating bar control that shows the progress (position) between a configurable minimum and maximum. It can be also be shown in marquee style, indicating that a process of indeterminate duration is busy.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"><PROGRESS></PROGRESS></DIV>
ElementID	UniqueID

Properties for TWebProgressBar

Max	Sets the maximum value of the progress bar
Min	Sets the minimum value of the progress bar
Position	Sets the position of the progress bar
Style	Sets the style of the progress bar: pbstNormal: normal progress bar style pbstMarquee: marquee progress bar style for processes of indeterminate duration pbstDIV: progressbar is made up of DIV elements that can be styled by Bootstrap

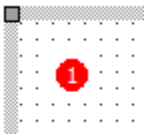

TWebBadge



Description

The TWebBadge is a badge control that can be used standalone or as part of other controls (like a TWebListControl, TWebTableControl, ...)

The badge can work standalone but is also designed so it can directly use Bootstrap styles.

 <p>Designtime</p>	 <p>Runtime</p>
---	--

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	
ElementID	UniqueID

Properties for TWebBadge

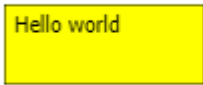
Color	Sets the background color of the badge
Text	Sets the text in the badge
TextColor	Sets the badge text color

TWebPaintBox



Description

Below is a list of the most important properties methods and events for the TWebPaintBox. TWebPaintBox is a group of radio button controls similar to a VCL TPaintBox.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

Properties for TWebPaintBox

ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file

Events for TWebPaintBox

OnPaint	Event triggered when the paintbox needs to be repainted. The WebPaintBox.Canvas can be used as in the VCL TPaintBox to draw within the paintbox control
OnTouchEnd	Event triggered when a touch on the paintbox ends

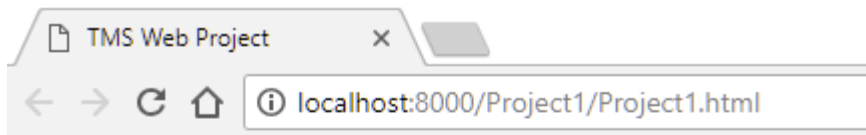
OnTouchMove	Event triggered when a move is made while touching the paintbox control
OnTouchStart	Event triggered when a touch on the paintbox starts

This example code snippet demonstrates how to paint something in the TWebPaintBox:

```
procedure TForm1.WebPaintBox1Paint(Sender: TObject);
begin
  WebPaintBox1.Canvas.Pen.Width := 3;
  WebPaintBox1.Canvas.Pen.Color := clRed;
  WebPaintBox1.Canvas.Brush.Color := clYellow;
  WebPaintBox1.Canvas.Brush.Style := bsSolid;
  WebPaintBox1.Canvas.Rectangle(10,10,250,100);

  WebPaintBox1.Canvas.Font.Name := 'Arial';
  WebPaintBox1.Canvas.Font.Size := 14;
  WebPaintBox1.Canvas.TextOut(20,40,'Painted by Pascal code');
end;
```

Result:



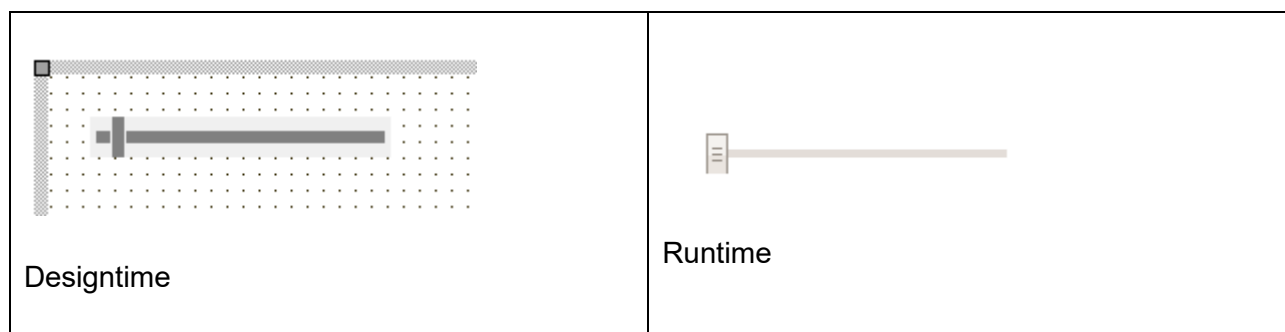
Painted by Pascal code

TWebTrackBar



Description

Below is a list of the most important properties methods and events for TWebTrackBar. TWebTrackBar is a trackbar control similar to a VCL TTrackBar. Note: in order to use the TWebTrackBar control, a fully HTML5 compliant browser is needed.



HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<INPUT TYPE="RANGE" ID="UniqueID">
ElementID	UniqueID

Properties for TWebTrackBar

ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing

	HTML element in the form HTML file
Max	Sets the maximum value of the trackbar
Min	Sets the minimum value of the trackbar
Position	Sets the thumb position of the trackbar
TabOrder	Sets the tab order of the control
TabStop	When true, the focus is turned to the control when pressing tab

Events for TWebTrackBar

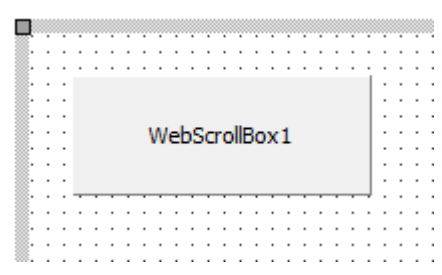
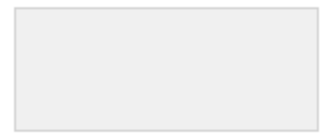
OnChange	Event triggered when the thumb on the trackbar is moved
----------	---

TWebScrollBox



Description

Below is a list of the most important properties methods and events for TWebScrollBox. TWebScrollBox is a container control that shows a scrollbar when it hosts child controls exceeding the client area of the control. TWebScrollBox is similar to a VCL TScrollBox.

 <p>WebScrollBox1</p> <p>Designtime</p>	 <p>Runtime</p>
--	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	
ElementID	UniqueID

Properties for TWebScrollBox

AutoScroll	When true, the scrollbar will be automatically displayed when child controls exceed the client rectangle of the scrollbox control
BorderStyle	Sets the border style of the scrollbox
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file

Events for TWebScrollBox

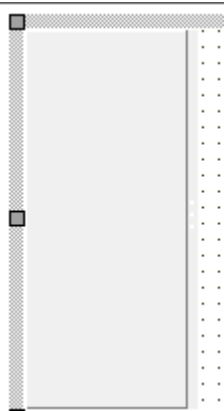
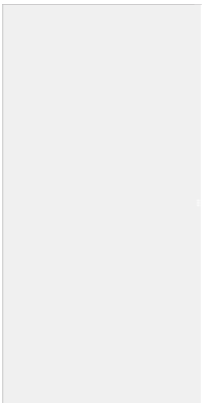
OnClick	Event triggered when the scrollbox is clicked
OnDbClick	Event triggered when the scrollbox is double-clicked

TWebSplitter



Description

Below is a list of the most important properties methods and events for TWebSplitter. TWebSplitter is a splitter control that allows to change sizes of other controls aligned on the form when the splitter is moved. TWebSplitter is similar to the VCL TSplitter.

	
Designtime	Runtime

Properties for TWebSplitter

ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
GripColor	Sets the color of the grip dots in the middle of the splitter control

Events for TWebSplitter

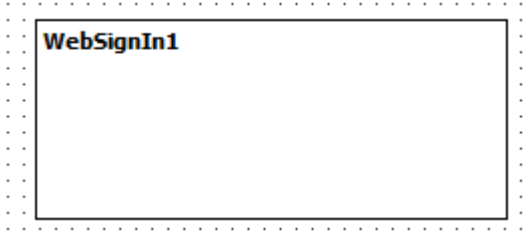
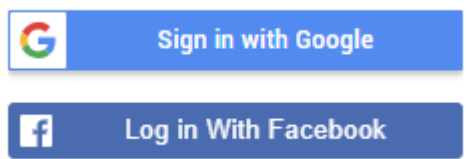
OnMoved	Event triggered when the splitter was moved by the user
---------	---

TWebSignIn



Description

Below is a list of the most important properties methods and events for TWebSignIn.
TWebSignIn allows letting users sign in through an existing Google or Facebook account.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebSignIn

ElementClassName	Optionally sets the CSS classname for the edit control when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML INPUT element in the form HTML file the edit control needs to be connected with. When connected, no new edit control is created but the Delphi class is connected with the existing HTML element in the form HTML file
Services: Google and Facebook	
Enabled	Sets if signin through the respective service is enabled
Visible	Sets if the signin button is visible. If Visible is False and Enabled is True you can still start the signin procedure programmatically (See SignIn method)
AppKey	Sets the API Key used to identify with the respective service. (See the topic "TWebSignIn usage" for information on how to obtain an API Key)
ControlID	Sets the ID of the HTML element where the signin button should be displayed. If the value is left empty the button is rendered inside the TWebSignIn control.

Methods for TWebSignIn

SignIn(Service)	Starts the signin procedure programmatically for the service specified with the Service parameter
SignOut(Service)	Starts the signout procedure programmatically for the service specified with the Service parameter

Events for TWebSignIn

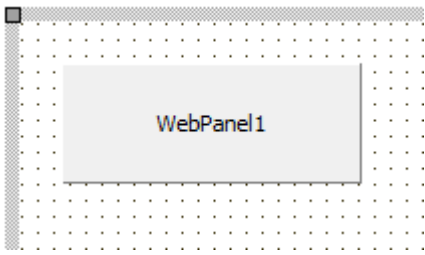
OnGoogleSignedIn	Event triggered if a user successfully signed in through Google. The event provides the user's Token, ID, FirstName, LastName, ImageUrl and Email via the Args parameter values.
OnGoogleSignedOut	Event triggered if a user successfully signed out through Google
OnFacebookSignedIn	Event triggered if a user successfully signed in through Facebook. The event provides the user's ID, Name and Email via the Args parameter values.
OnFacebookSignedOut	Event triggered if a user successfully signed out through Facebook

TWebPanel



Description

Below is a list of the most important properties methods and events for TWebPanel. TWebPanel is a container control that can host other child controls. TWebPanel is similar to a VCL TPanel.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebPanel

AutoSize	When true, the size of the panel automatically adapts to space the child controls it contains takes.
BorderStyle	Sets the style of the border
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file

Events for TWebPanel

OnClick	Event triggered when the panel is clicked
OnDbClick	Event triggered when the panel is double-clicked

TWebHTMLContainer



Description

Below is a list of the most important properties methods and events for TWebHTMLContainer. TWebHTMLContainer is basically a placeholder to add any HTML to be rendered on the page directly on the form. The HTML is added as text via the property WebHTMLContainer.HTML. The outer element of the HTML container is a DIV element.

<p>Designtime</p>	<p>Sample HTML content here</p> <p>Runtime</p>
-------------------	--

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	
ElementID	UniqueID

Properties for TWebHTMLContainer

HTML	A stringlist holding the HTML (as text) that will be rendered in a DIV
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the

	Delphi class is connected with the existing HTML element in the form HTML file
--	--

Events for TWebHTMLContainer

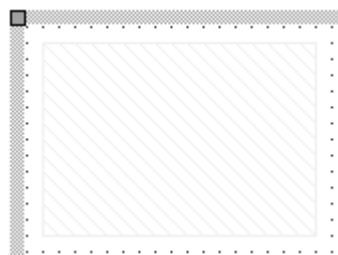
OnClick	Event triggered when the outer DIV of the HTML container is clicked
OnDbClick	Event triggered when the outer DIV of the HTML container is double-clicked
OnMouseDown	Event triggered when the outer DIV of the HTML container is clicked
OnMouseUp	Event triggered when the mouse goes up on the outer DIV of the HTML container
OnMouseMove	Event triggered when the mouse moves over the outer DIV of the HTML container

TWebHTMLForm



Description

TWebHTMLForm is just a structural control that represents the FORM HTML element as structural element for the INPUT controls it has. The TWebHTMLForm is needed to indicate a section of INPUT controls on the page and it will be rendered as `<FORM> ... child controls here </FORM>`

 <p>Design-time</p>	<p>The HTML form is a structural element and not visible at runtime. Controls in the form are positioned on the form as if the TWebHTMLForm does not exist.</p> <p>Runtime</p>
--	--

--	--

HTML template tag

The HTML tag the component can be associated with a FORM element in a HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<FORM ID="UniqueID"></FORM>
ElementID	UniqueID

Properties for TWebHTMLForm

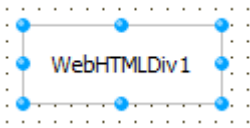
Name	Sets the name of the HTML FORM element
------	--

TWebHTMLDiv



Description

TWebHTMLDiv is just a structural control that represents the DIV HTML element.

 <p>Designtime</p>	<p>The TWebHTMLDIV is a control that represents a HTML DIV element</p> <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with a DIV element in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebHTMLDiv

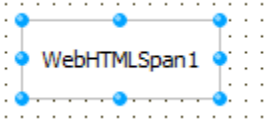
ElementClassName	Sets the CSS class name attributes
ElementFont	Determines whether the Font property values will be applied as font style or if CSS based font settings will be used
ElementPosition	Defines whether the DIV is shown absolute positions or relative positioned
HTML: THTMLText	Sets the innerHTML text value for the DIV element
Name	Sets the name of the HTML DIV element

TWebHTMLSpan



Description

TWebHTMLSpan is just a structural control that represents the SPAN HTML element.

 <p>Designtime</p>	<p>The TWebHTMLSpan is a control that represents a HTML SPAN element</p> <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with a SPAN element in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	
ElementID	UniqueID

Properties for TWebHTMLSpan


ElementClassName	Sets the CSS class name attributes
ElementFont	Determines whether the Font property values will be applied as font style or if CSS based font settings will be used
ElementPosition	Defines whether the SPAN is shown absolute positions or relative positioned
HTML: THTMLText	Sets the innerHTML text value for the SPAN element
Name	Sets the name of the HTML SPAN element

TWebHTMLAnchor



Description

TWebHTMLAnchor is just a structural control that represents the ANCHOR HTML element <A>.

 <p>Designtime</p>	<p>The TWebHTMLAnchor is a control that represents a HTML ANCHOR element</p> <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with a ANCHOR element in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	
ElementID	UniqueID

Properties for TWebHTMLAnchor

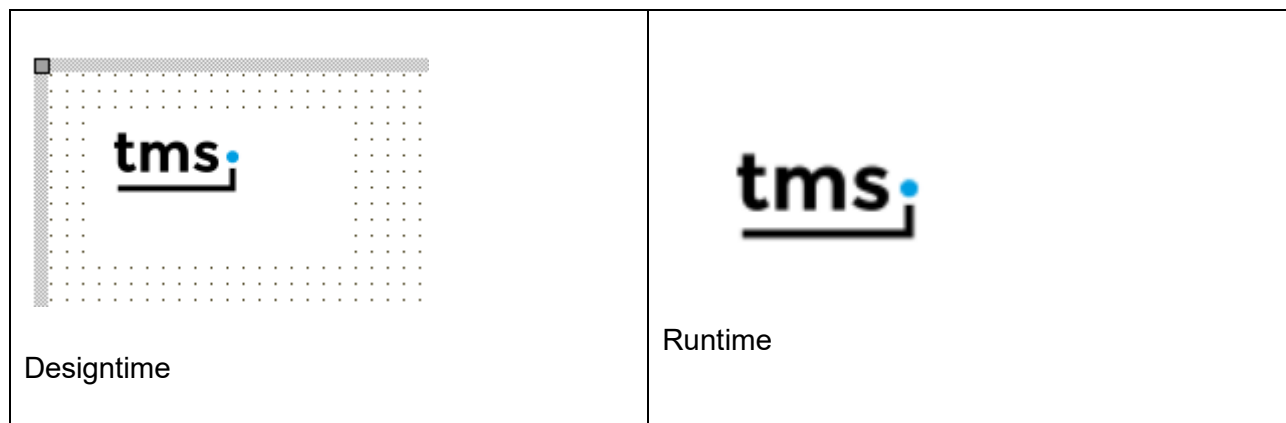
ElementClassName	Sets the CSS class name attributes
ElementFont	Determines whether the Font property values will be applied as font style or if CSS based font settings will be used
ElementPosition	Defines whether the ANCHOR is shown absolute positions or relative positioned
HTML: THTMLText	Sets the innerHTML text value for the ANCHOR element
Name	Sets the name of the HTML ANCHOR element

TWebImageControl



Description

Below is a list of the most important properties methods and events for TWebImageControl. TWebImageControl can display an image on the form. TWebImageControl is similar to a VCL TImage.



HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	
ElementID	UniqueID

Properties for TWebImageControl

AutoSize	When true, the size of the control automatically adapts to the size of the image it contains
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used

ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
Picture	Sets the picture at design time. Note that the image is automatically deployed by the compiling process to a separate image file with a unique name.
URL	Specifies the image as an URL

Methods for TWebImageControl

DataURL: string	Returns the image control image content as data URL string
DataURL(Width,Height: integer): string;	Returns the image control image content at size width/height as data URL string

Events for TWebImageControl



OnClick	Event triggered when the image is clicked
OnDbClick	Event triggered when the image is double-clicked
OnLoaded	Event triggered when the image load completed after assigning the URL or DataURL

TWebImageZoomControl



Description

Below is a list of the most important properties methods and events for TWebImageZoomControl. TWebImageZoomControl can display an image on the form and display a zoomed-in image version in a popup when it is clicked.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	
ElementID	UniqueID

Properties for TWebImageControl

Appearance	
HeightPercent	Sets the percent of the browser window height that is taken up by the popup
ResponsiveHeightPercent	Sets the percent of the browser window height that is taken up by the popup when

	the available browser window width is equal or less than ResponsiveMaxWidth
ResponsiveWidthPercent	Sets the percent of the browser window width that is taken up by the popup when the available browser window width is equal or less than ResponsiveMaxWidth
ResponsiveMaxWidth	Sets the maximum browser window width for the ResponsiveHeightPercent and ResponsiveWidthPercent values are used, otherwise the HeightPercent and WidthPercent values are used
WidthPercent	Sets the percent of the browser window width that is taken up by the popup
AutoSize	When true, the size of the control automatically adapts to the size of the image it contains
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
Picture	Sets the picture at design time. Note that the image is automatically deployed by the compiling process to a separate image file with a unique name.
PictureZoom	Sets the zoom picture at design time. Note that the image is automatically deployed by the compiling process to a separate image file with a unique name.
URL	Specifies the image as an URL
URLZoom	Specifies the zoom image that is displayed when the image is clicked as an URL

Events for TWebImageControl

OnClick	Event triggered when the image is clicked
OnDbClick	Event triggered when the image is double-clicked

TWebLinkLabel



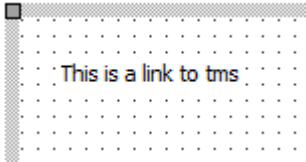
Description

Below is a list of the most important properties methods and events for TWebLinkLabel. TWebLinkLabel is similar to a VCL TLinkLabel.

For a sample TWebLinkLabel with caption set to:

This is a link to `tms`

the result is:

 <p>Designtime</p>	<p>This is a link to tms</p> <p>Runtime</p> <p>Runtime</p>
---	--

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebLinkLabel

AutoSize	When true, the size of the label control automatically adapts to the text it contains
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
EllipsisPosition	Sets the type of ellipsis to use for showing the text when it doesn't fit in the label rectangle. epNone: no ellipsis used epEndEllipsis: ellipsis at the end of the text epPathEllipsis: label text contains a path name and ellipsis is set taking a file path in account epWordEllipsis: ellipsis is positioned at word boundary
Layout	Sets the vertical text position in the label tlTop: top aligned tlCenter: center aligned tlBottom: bottom aligned
WordWrap	When true, the text can be displayed wordwrapped in the label client rect

Events for TWebLinkLabel

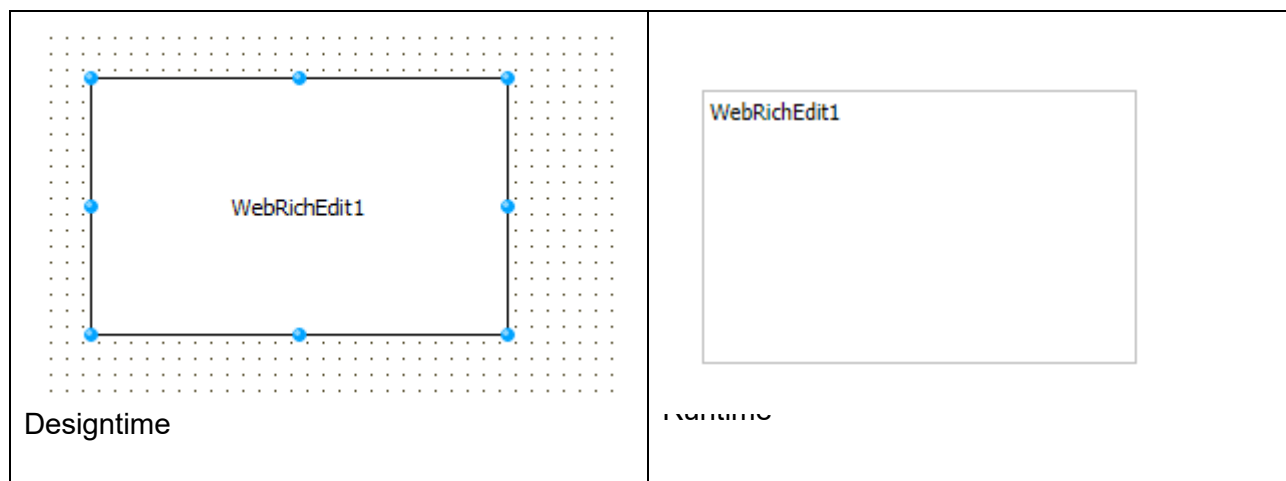
OnClick	Triggered when the label is clicked
OnDbClick	Triggered when the label is double-clicked
OnLinkClick	Event triggered when a hyperlink in the TWebLinkLabel is clicked

TWebRichEdit



Description

Below is a list of the most important properties methods and events for TWebRichEdit. TWebRichEdit is a control that allows to edit text and apply text formatting. TWebRichEdit is similar to a VCL TRichEdit.



HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebRichEdit

AutoSize	
BorderStyle	Sets the border style
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a

	HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
PlainText	Gets or sets the text of the rich editor control as plain text
SelAttributes	Gets or sets the attributes of the selected text in the rich editor control
Text	Gets or sets the text of the rich editor control as HTML formatted text

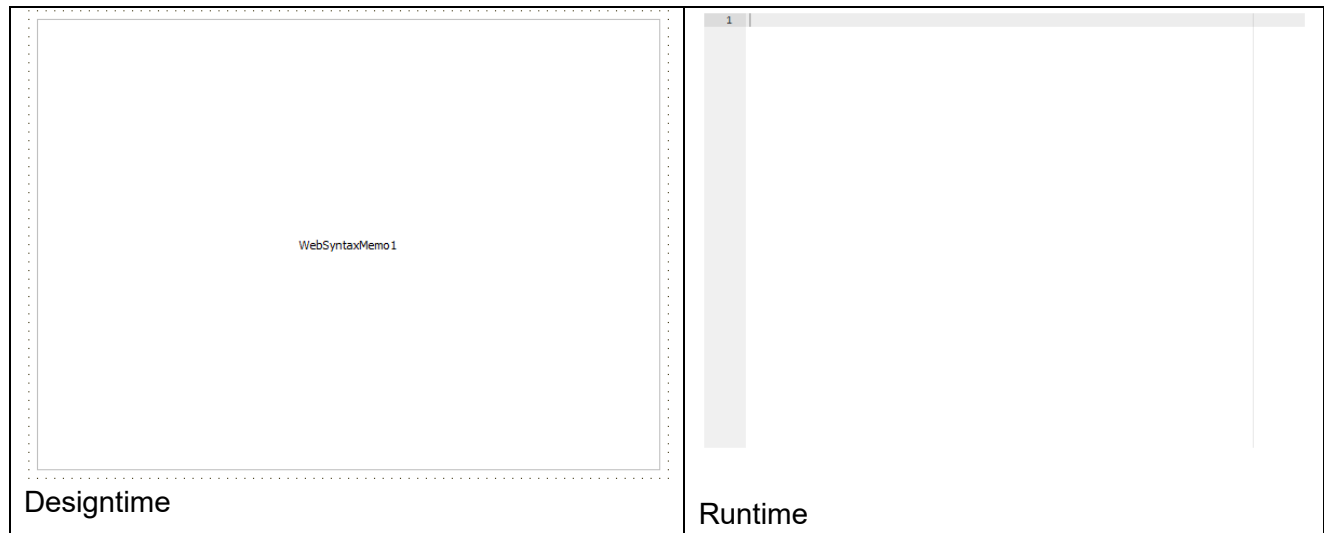
Events for TWebRichEdit

OnClick	Triggered when the rich editor is clicked
OnDbClick	Triggered when the rich editor is double-clicked
OnSelectionChange	Triggered when the selection within the rich editor is changed

TWebSyntaxMemo



Below is a list of the most important properties methods and events for the TWebSyntaxMemo. TWebSyntaxMemo is using the external JavaScript written [Ace](#) editor.



Loading a file

Loading with TWebFilePicker

First the file should be retrieved as a text. This can be done in the TWebFilePicker's OnChange event:

```
procedure TForm4.WebFilePicker1Change(Sender: TObject);
begin
    //First make sure that there's a file available
    if Assigned(WebFilePicker1.Files[0]) then
    begin
        WebFilePicker1.Files[0].GetFileAsText;
        //Additional code here
    end;
end;
```

Then assign the retrieved text to the TWebSyntaxMemo:

```
procedure TForm4.WebFilePicker1GetFileAsText(Sender: TObject;
    AFileIndex: Integer; AText: string);
begin
    WebSyntaxMemo1.Text := AText;
end;
```

Loading with drag and drop

For this approach a TWebFileReader is needed. Once the file is readed, the text content can be

assigned to the TWebSyntaxMemo.

```
procedure TForm4.WebFormCreate(Sender: TObject);
begin
    fr := TWebFileReader.Create(Self);
    fr.OnReadDone := DoReadLoaded;
end;

procedure TForm4.DoReadLoaded(aFileName: string; AResult: JSValue);
begin
    WebSyntaxMemo1.Text := JS.toString(AResult);
end;
```

What's left to handle is the file reading itself when a file has been dropped onto the TWebSyntaxMemo. In the OnDragDrop event, the following can be written:

```
procedure TForm4.WebSyntaxMemo1DragDrop(Sender, Source: TObject; X, Y:
Integer);
var
    f: TJSHTMLFile;
begin
    f := TJSDragEvent(TDragSourceObject(Source).Event).dataTransfer.files[0];
    //Get the file
    //Make sure it's available
    if Assigned(f) then
        fr.readAsBinaryString(f); //Read the file using the TWebFileReader
end;
```

Downloading a file

Downloading a file means a single line of code only.

For example with the code below, the contents of the editor can be downloaded to the test.txt file.

```
Application.DownloadTextFile(WebSyntaxMemo1.Text, 'test.txt');
```

Properties for TWebSyntaxMemo

Property	Description
Autocompletion	There are 3 options: <code>saNone</code> to disable autocompletion, <code>saLive</code> to autocomplete during typing and <code>saBasic</code> to show autocompleting keywords by pressing <code>Ctrl+Space</code> .
CaretPosition: Integer	Position of the caret.
CustomAutocomplete	A collection of custom keywords that can be added to autocollection. Keyword highlighting is not available.

FadeFoldWidgets: Boolean	Enable or disable fading fold widgets.
FixedGutterWidth: Boolean	Gutter width can be fixed up to 1000 lines.
FontName: string	Name of the font. Only monospaced fonts will work.
FontSize: Integer	Size of the font.
HighlightActiveLine: Boolean	Highlight the line where the caret is.
Lines: TStringList	Access the editor's content as a TStringList.
Mode: TSyntaxMemoMode	Language mode for the editor.
PersistentHorizontalScrollbar: Boolean	Always show horizontal scrollbar.
PersistentVerticalScrollbar: Boolean	Always show vertical scrollbar.
PrintMargin: Integer	Value of the print margin position. Default is 80.
ReadOnly: Boolean	Enable or disable read only mode.
SelLength: Integer	Selection length.
SelStart: Integer	Selection start.
ShowFoldWidgets: Boolean	Hide or show the fold widgets.
ShowGutter: Boolean	Hide or show the gutter.
ShowIndentGuides: Boolean	Hide or show the indent guides.
ShowInvisibles: Boolean	Hide or show the invisible characters such as whitespaces.
ShowLineNumbers: Boolean	Hide or show the line numbers.
ShowPrintMargin: Boolean	Hide or show the print margin.
SoftTabs: Boolean	Enable or disable soft tabs.
TextDirection: TSyntaxTextDirection	Text direction from left to right or right to left.
TabSize: Integer	Size of the tab in spaces.
Text: string	Access the editor's content as a single string.
Theme: TSyntaxMemoTheme	The theme of the editor.
WordWrap	There are 4 options for wordwrapping: <code>swNone</code> means there's no wordwrap, <code>swPrintMargin</code> will wrap at the print margin, <code>swView</code> will wrap at what's visible and <code>swValue</code> will use the <code>WordWrapValue</code> to wrap at a configured length.
WordWrapIndented: Boolean	Allow indenting in wordwrap.
WordWrapValue: Integer	Wordwrap size. Only used if the <code>WordWrap</code> is set to <code>swValue</code>

Methods for TWebSyntaxMemo

Property	Description
Clear	Clears the content of the editor.

DisableLocalKeywords	Disables local keywords that are added constantly while content is being added to the editor.
Find(AText: string)	Finds and highlights the AText (if exists) in the editor's content.
FindAll(AText: string)	Finds all and highlights the first AText (if exists) in the editor's content.
FindNext	Finds the next occurrence of the highlighted text.
FindPrevious	Finds the previous occurrence of the highlighted text.
Focus	Focuses the editor.
InitializeKeyWords(ACompleter: TSyntaxCompleter)	Initialize a set of keywords with ACompleter.
InsertText(AText: string)	Insert AText at the caret position.
InsertText(APosition: TPoint; AText: string)	Insert AText at APosition.
OpenSearchBox	Opens the editor's searchbox.
PreloadPascalKeywords	Earlier versions of Ace does not support Pascal keywords in autocompletion. With this function, they can be preloaded as autocompletion keywords.
Redo	'Redo' edit command. Redoes an undid change.
RemoveSelectedText	Remove the selected text.
RemoveCustomAutocompleter	Remove the added custom autocompleter.
RemovePascalKeywords	Earlier versions of Ace does not support Pascal keywords in autocompletion. With this function, they can be removed from autocompletion keywords if they had been added previously.
Replace(AReplacement: string)	'Replace' edit command. It replaces the selected text with AReplacement.
Replace(AText, AReplacement: string)	'Replace' edit command. It replaces AText with AReplacement.
ReplaceAll(AReplacement: string)	'Replace all' edit command. It replaces all occurrences of the selected text with AReplacement.
ReplaceAll(AText, AReplacement: string)	'Replace all' edit command. It replaces all occurrences of AText with AReplacement.
SelectAll	'Select all' edit command. Selects all of the text.
Undo	'Undo' edit command. Undoes the previous change.
Unselect	'Unselect' edit command. Unselects everything.

Events for TWebSyntaxMemo


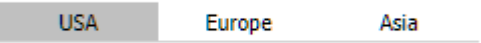
Property	Description
OnChangeCursor	Event triggered when cursor position has changed.
OnChangeSelection	Event triggered when text selection has changed.
OnDragDrop	Event triggered when something is dropped onto the editor.
OnDragOver	Event triggered when something is dragged over the editor.

TWebTabSet



Description

Below is a list of the most important properties methods and events for TWebTabSet.
TWebTabSet is similar to a VCL TTabSet.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	
ElementID	UniqueID

Properties for TWebTabSet

ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
ItemIndex	Sets or gets the selected tab
Items	List of tab captions

SelectedColor	Sets the background color of the selected tab
---------------	---

Methods for TWebTabSet

Clear	Removes all tabs
SelectNextTab	Selects the next or previous page in the page control, depending on the value of the parameter.

Events for TWebTabSet

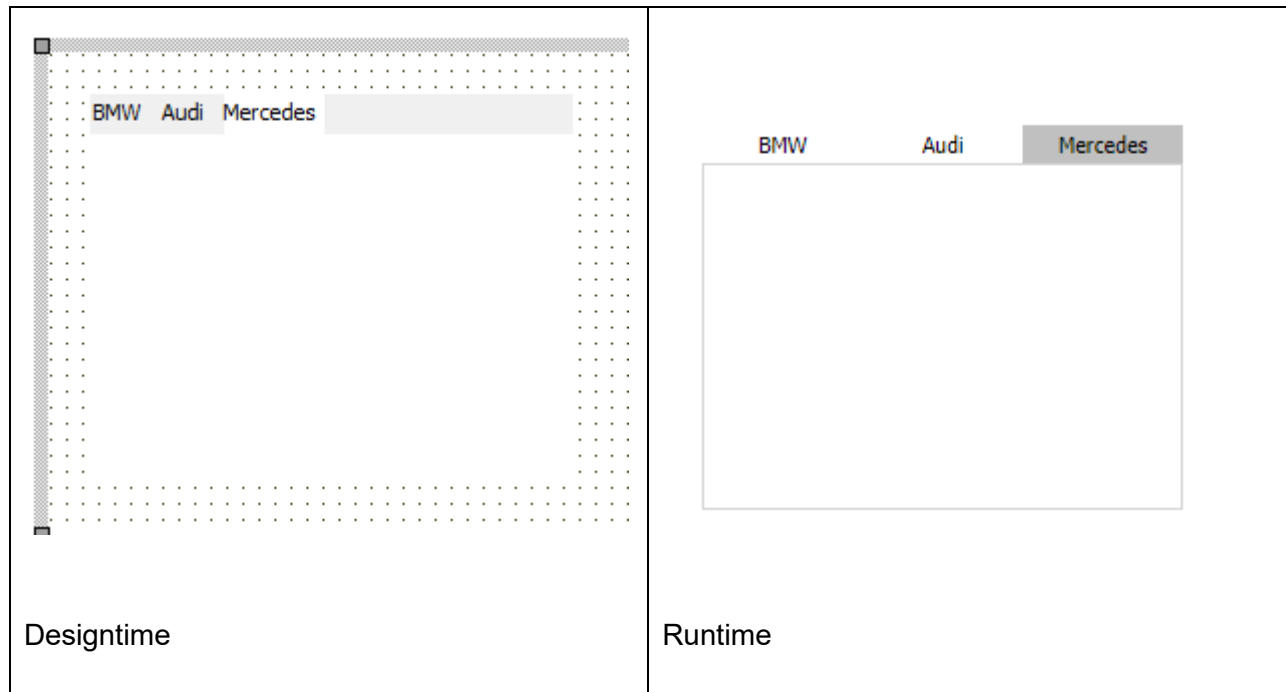
OnClick	Event triggered when a tab is clicked
OnDbClick	Event triggered when a tab is double-clicked
OnSelectionChange	Event triggered when the selected tab changes

TWebPageControl



Description

Below is a list of the most important properties methods and events for TWebPageControl. TWebPageControl is similar to a VCL TPageControl.



HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	
ElementID	UniqueID

Make sure to set at least a width/height for the outer span as the tabsheet HTML elements in the pagecontrol are set in the outer container element as absolute positioned.

Properties for TWebPageControl

ActivePage: TWebTabSheet	Gets or sets the active page in the page control
ActivePageIndex: integer	Gets or set the active page by its index
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the

	label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
PageCount: integer	Returns the number of pages
Pages[Index: integer]: TWebTabSheet	Provides access to the pages in the page control
SelectedColor: TColor	Sets the background color of the selected tab
SelectedTextColor: TColor	Sets the text color of the selected tab
ShowTabs: boolean	When true, the tabs of the page control are visible
TabIndex	Sets or gets the selected page index

Methods for TWebPageControl

SelectNextPage	Selects the next or previous page in the page control, depending on the value of the parameter.
----------------	---

Events for TWebPageControl

OnChange	Event triggered when the active page of the page control changes
OnClick	Event triggered when the page is clicked
OnDbClick	Event triggered when the page is double-clicked

TWebTabSheet



Description

TWebTabSheet is the container control used in a TWebPageControl to host controls on a sheet.

TWebLoginPanel



Description

TWebLoginPanel is a control designed to capture a user email and login code for sign-in in a web application.

Designtime	Runtime

Properties for TWebLoginPanel

CaptionLabel	Sets the caption of the loginpanel
Color	Sets the background color of the loginpanel
ElementButtonClassName	Optionally sets the CSS classname for the button in the login panel when styling via CSS is used
ElementCaptionClassName	Optionally sets the CSS classname for the caption in the login panel when styling via CSS is used
ElementClassName	Optionally sets the CSS classname for the login panel when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the

	label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
ElementInputClassName	Optionally sets the CSS classname for the input controls in the login panel when styling via CSS is used
ElementLabelClassName	Optionally sets the CSS classname for the labels in the login panel when styling via CSS is used
LoginLabel	Sets the caption for the login panel
PasswordLabel	Sets the caption for the label to indicate the password input field
UserLabel	Sets the caption for the label to indicate the username input field

Events for TWebLoginPanel



OnClick	Event triggered when the panel is clicked
OnLogin	Event triggered when the login button is clicked

TWebSpeedButton



Description

Below is a list of the most important properties methods and events for TWebSpeedButton. TWebSpeedButton is similar to a VCL TSpeedButton.

 Design-time	 Runtime
--	--

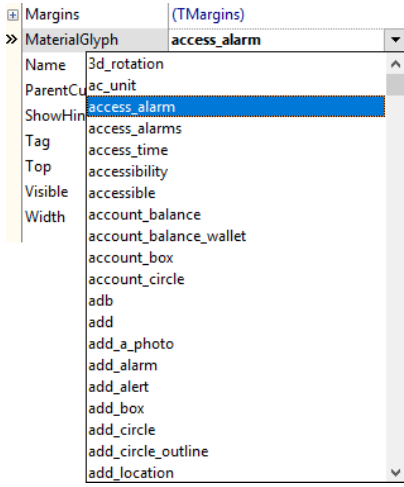
HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<BUTTON ID="UniqueID"></BUTTON>
ElementID	UniqueID

Properties for TWebSpeedButton

AllowAllUp	When there is a group of speed buttons, depending on AllowAllUp, there is always a button down or not
Caption	Sets the speedbutton text
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a

	HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
Enabled	Sets the button as enabled or disabled
Flat	When true, the button is displayed in flat style
Glyph	Sets the image for the speed button
GroupIndex	To group buttons, set the GroupIndex indential for multiple speed buttons
MaterialGlyph	Allows to pick an icon from the Google material icon set 
MaterialGlyphColor	Sets the color of the material glyph icon
MaterialGlyphSize	Sets the size of the material glyph icon

Events for TWebSpeedButton



OnClick	Event triggered when the speed button is clicked
OnDbClick	Event triggered when the speed button is double-clicked

TWebPayPal



Description

Below is a list of the most important properties methods and events for TWebPayPal.
TWebPayPal allows using the PayPal checkout process.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebPayPal

APIKey	<p>Sets the API key retrieved from the PayPal developers console Dashboard at: https://developer.paypal.com/</p> <p>Full instructions can be found here.</p> <p>Notes:</p> <ul style="list-style-type: none"> - The value required for the API Key is referred to as "Client ID" on The PayPal
--------	---

	<p>Dashboard.</p> <ul style="list-style-type: none"> - The PayPal button will only be displayed if an API key is provided. - The API key can not be changed once the PayPal button has been initialized.
<p>Payment Configure the PayPal payment details before the user can initiate the payment process.</p>	
Address1	Sets line 1 of the payer address
Address2	Sets line 2 of the payer address (optional)
City	Sets the city of the payer address
CountryCode	<p>Sets the country code of the payer address</p> <p>Note: If the country code is set to US (United States) or CA (Canada) a valid value is required in the City, PostalCode and State property</p>
Currency	<p>Sets the currency of the PayPal payment</p> <p>Note: The Currency can not be changed once the PayPal button has been initialized.</p>
CustomText	Sets a custom text to include with the PayPal payment (optional)
Description	Sets the description text associated with the PayPal payment (optional)
HandlingFee	Sets the handling fee cost associated with the PayPal payment (optional)
Insurance	Sets the insurance cost associated with this PayPal payment (optional)
InvoiceNumber	Sets the invoice number associated with the PayPal payment (optional, must be unique)
<p>Items Collection of items associated with the PayPal payment</p>	
Description	Sets the description of the item
Name	Sets the name of the item
Price	Sets the price of the item
Quantity	Sets the number of items
SKU	Sets the SKU associated with the item

Tag	Sets the tag associated with the item (optional)
TagObject	Sets the object associated with the item (optional)
Tax	Sets the tax cost associated with the item (optional)
Locale	Sets the language used in the PayPal checkout interface Note: The Locale can not be changed once the PayPal button has been initialized.
Phone	Sets the phone number of the payer (optional)
PostalCode	Sets the postal code of the payer address
RecipientName	Sets the name of the payer (optional)
Shipping	Sets the shipping cost associated with the PayPal payment (optional)
ShippingDiscount	Sets the shipping cost discount associated with the PayPal payment (optional)
State	Sets the state of the payer address (optional, except if CountryCode is set to US or CA)
Tax	Sets the tax cost associated with the PayPal payment (optional)

Events for TWebPayPal

OnPaymentDone	
Event triggered when a PayPal payment was executed successfully.	
Arguments:	
Address1	Line 1 of the payer address
Address2	Line 2 of the payer address
City	The city of the payer address
CountryCode	The country code of the payer address
Currency	The currency associated with the PayPal

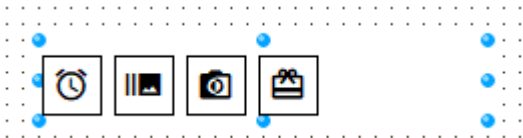
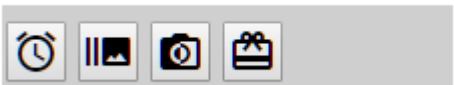
	payment
CustomText	The custom text associated with the PayPal payment
Description	The payment description
Email	The email address of the payer
FirstName	The first name of the payer
InvoiceNumber	The invoice number associated with the PayPal payment
LastName	The last name of the payer
OrderID	The order ID associated with the PayPal payment
PayerID	The payer ID associated with the PayPal payment
PaymentID	The payment ID associated with the PayPal payment
PaymentState	The state of the PayPal payment
Phone	The phone number of the payer
PostalCode	The postal code of the payer address
RecipientName	The name associated with the shipping address
SaleID	The sale ID associated with the PayPal payment
State	The state of the payer address
Total	The total cost of the PayPal payment
OnPaymentCancelled	Event triggered when a PayPal payment was cancelled by the user
OnPaymentError Event triggered when an error occurred during the PayPal checkout process.	
Arguments:	
ErrorName	The name of the error that occurred
ErrorDetails	A list of details about the error

TWebToolBar



Description

Below is a list of the most important properties methods and events for TWebToolBar. A TWebToolBar is a container control that can host several controls to form a toolbar.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

Properties for TWebToolBar

ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file

Events for TWebToolBar

OnClick	Event triggered when the toolbar is clicked
OnDbClick	Event triggered when the toolbar is double-clicked

Example: line wrapping (responsive behaviour)

To make the content of the TWebToolBar automatically adapt to the available width, set the WidthStyle and HeightStyle of the TWebToolBar to ssAuto. Also set the ElementPosition of each component contained in the TWebToolBar to epRelative.

The order in which the components are displayed can be controlled with the ChildOrder property.

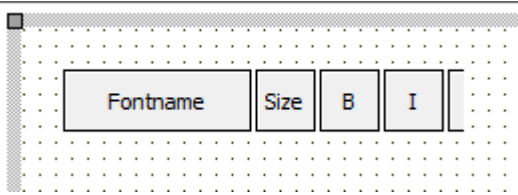

```
WebSpeedButton1.ElementPosition := epRelative;  
WebSpeedButton1.ChildOrder := 0;  
WebSpeedButton2.ElementPosition := epRelative;  
WebSpeedButton2.ChildOrder := 1;  
WebToolBar1.WidthStyle := ssAuto;  
WebToolBar1.HeightStyle := ssAuto;
```

TWebRichEditToolbar



Description

Below is a list of the most important properties methods and events for TWebRichEditToolbar.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

Properties for TWebRichEditToolbar

ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
Hints	Contains the list of hint property values for the buttons in the ribbon
RichEdit	Sets the TWebRichEdit component with which the toolbar interacts
VisibleButtons	Sets what button on the toolbar are visible. This is a set property with following possible values: reFont, reFontSize, reBold, reItalic, reUnderline, reStrikeThrough, reAlignLeft, reAlignCenter, reAlignRight, reUnorderedList, reOrderedList,

	reForegroundColor, reBackgroundColor
--	--------------------------------------

Events for TWebRichEditToolbar

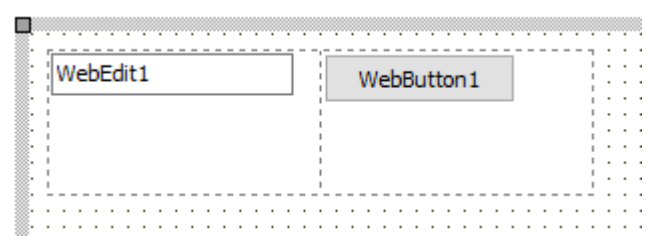

OnClick	Event triggered when the toolbar is clicked
OnDbClick	Event triggered when the toolbar is double-clicked

TWebGridPanel



Description

Below is a list of the most important properties methods and events for TWebGridPanel.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	
ElementID	UniqueID

Properties for TWebGridPanel

ColumnCollection	<p>Access to the collection of columns in the grid panel. The width, alignment, CSS, width style of each column can be specified</p> <p>Alignment: sets the vertical alignment in the row to taLeftJustify, taCenter, taRightJustify ElementClassName: sets an optional CSS class name for the column MarginLeft: sets a left margin in pixels MarginRight: sets a right margin in pixels SizeStyle: sets the style of the width specification as percent, absolute, auto Value: sets the width value</p>
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
ExpandStyle	<p>Can be set to</p> <p>esAddRows: new rows are added when new controls are inserted an no more grid cells are available esAddColumns: new columns are added when new controls are inserted an no more grid cells are available</p>
GridLineColor	Sets the color of the grid lines
GridLineWidth	Sets the width of the grid lines
RowCollection	<p>Access to the collection of rows in the grid panel. The height, alignment, CSS, height style of each row can be specified</p> <p>Alignment: sets the vertical alignment in the</p>

	row to vaTop, vaCenter, vaBottom ElementClassName: sets an optional CSS class name for the row MarginBottom: sets a bottom margin in pixels MarginTop: sets a top margin in pixels SizeStyle: sets the style of the height specification as percent, absolute, auto Value: sets the height value
--	---

Events for TWebGridPanel

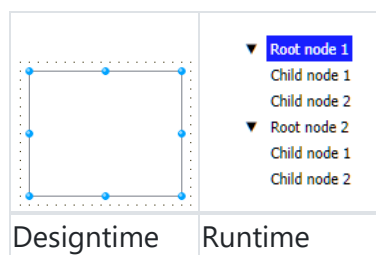
OnClick	Event triggered when the panel is clicked
OnDblClick	Event triggered when the panel is double-clicked

TWebTreeview



Description

Below is a list of the most important properties methods and events for TWebTreeview. TWebTreeview is similar to a VCL TTreeView.



HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebTreeview

Property	Description
AutoExpand	When true, a click on a node will select the node but also expand the child nodes.
ElementClassName	Optionally sets the CSS classname for the out DIV element of the treeview when styling via CSS is used
ElementNodeClassName	Optionally sets the CSS classname for the node SPAN element that is used for each node
ElementNodeSelectedClassName	Optionally sets the CSS classname for the node SPAN element that is used for each node when it is in selected state
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
Items	Hierarchical collection of nodes in the treeview. The interface to access nodes is similar to a the VCL nodes collection.
Selected	Gets or sets the selected TTreeNode in the treeview.

Methods for TWebTreeview

Property	Description
GetNodeElement	Gets the HTML element that is the container for the TTreeNode.
GetNodeFromID	Gets the TTreeNode from the ID of the HTML element.

Events for TWebTreeview

Property	Description
OnChange	Event triggered when the selected node in the treeview changed
OnChanging	Event triggered when the selected node in the treeview is about to change. The Allow parameter can be used to control if the selected node can change
OnClick	Event triggered when the control is clicked.
OnClickNode	Event triggered when a TTreeNode is clicked.

Property	Description
OnCollapsed	Event triggered when a node was collapsed.
OnCollapsing	Event triggered when a node is about to be collapsed. The Allow parameter can be used to control whether the node can be collapsed.
OnDbClick	Event triggered when the control is double-clicked.
OnDbClickNode	Event triggered when a TTreeNode is double-clicked.
OnExpanded	Event triggered when a node was expanded.
OnExpanding	Event triggered when a node is about to be expanded. The Allow parameter can be used to control whether the node can be expanded.
OnRenderNode	Event triggered when a node is about to be rendered. This returns a reference the HTML element that is the container for the node and allows further customization of the node.

Sample code

This code snippet shows how to programmatically add items to the treeview (very similar as with a VCL TTreeView)

```
var
    tn: TTreeNode;
begin
    WebTreeView1.BeginUpdate;

    tn := WebTreeView1.Items.Add('Root node 1');
    WebTreeView1.Items.AddChild(tn, 'Child node 1');
    WebTreeView1.Items.AddChild(tn, 'Child node 2');

    tn := WebTreeView1.Items.Add('Root node 2');
    WebTreeView1.Items.AddChild(tn, 'Child node 1');
    WebTreeView1.Items.AddChild(tn, 'Child node 2');

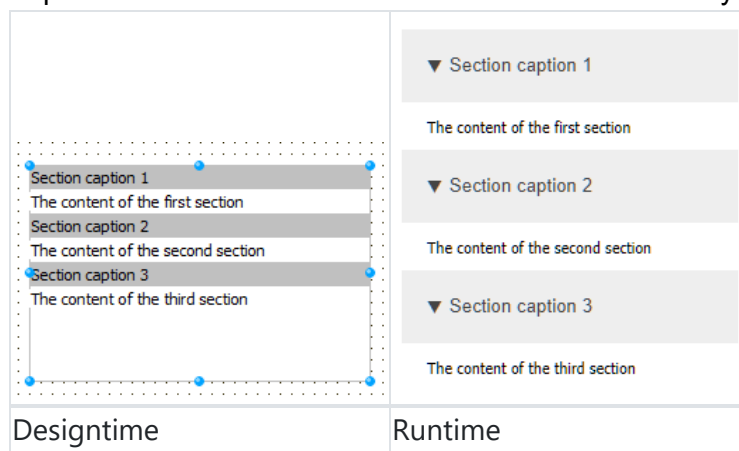
    WebTreeView1.EndUpdate;
end;
```

TWebAccordion



Description

Below is a list of the most important properties methods and events for TWebAccordion. An accordion is a collection of expandable sections. The sections are expanded by clicking a caption. The content of the section can be HTML or any other web controls.



HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebAccordion

Property	Description
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file

Property	Description
Sections	Collection of sections in the TWebAccordion

Events for TWebAccordion

Property	Description
OnCollapsed	Event triggered when a section was collapsed.
OnCollapsing	Event triggered when a section is about to be collapsed. The Allow parameter can be used to control whether the section can be collapsed.
OnExpanded	Event triggered when a section was expanded.
OnExpanding	Event triggered when a section is about to be expanded. The Allow parameter can be used to control whether the section can be expanded.
OnRenderSection	Event triggered when a section is about to be rendered. This returns a reference the HTML element that is the container for the section and allows further customization of the section.

Properties for TAccordionSection

Property	Description
Caption	Gets or sets the text or HTML of the section caption.
CaptionElement	Gets the HTML container element of the section caption.
Content	Gets or sets the text or HTML content of the section.
ContentElement	Gets the HTML container element of the section content.
Expanded	Gets or sets the expanded state of the section.
Tag	Integer property.

TWebResponsiveGridPanel



Description

Below is a list of the most important properties methods and events for TWebResponsiveGridPanel. TWebResponsiveGridPanel is grid panel with responsive behavior. This means that the layout of the grid panel can adapt to the form factor of the web page where it is used. This layout is controlled by the Layout collection. Like a regular grid panel, controls can be dropped on the TWebResponsiveGridPanel and these controls are organized in a grid like structure and represented and accessible via the TWebResponsiveGridPanel.ControlCollection.



HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebResponsiveGridPanel

Property	Description
ControlCollection	Collection of child controls of the TWebResponsiveGridPanel
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is

Property	Description
	created but the Delphi class is connected with the existing HTML element in the form HTML file
Layout	Collection of layout settings for various form factors. These layout settings are managed by a TResponsiveLayoutItem class. When a TWebResponsiveGridPanel is dropped on the form 4 layouts are automatically added for 4 different form factors: Smartphone with one column (screen width <= 575pixels), Tablet with two columns (screen width <= 768pixels), Desktop with three columns (screen width <= 991pixels), Large Desktop with four columns (screen width <= 1199pixels).

Methods for TWebResponsiveGridPanel

Property	Description
AddControl	Adds a new control to the TWebResponsiveGridPanel.
RemoveControl	Removes a new control from the TWebResponsiveGridPanel.

Events for TWebResponsiveGridPanel

Property	Description
OnLayoutChange	Event triggered when the form size changes and causes a new layout to be selected

TResponsiveLayoutItem is the class used in the Layout collection of the TWebResponsiveGridPanel to manage different desired layout settings per screen width.

Properties for TResponsiveLayoutItem

Property	Description
ColumnGap	Gets or sets the column gap in pixels (px) or percentage (%) for the layout. The column gap is the gap between two successive columns.
Description	Text property that can be used to describe the layout. The Description property is not used at runtime in the control.
Margins	Sets the margins of the responsive grid cells in the selected layout.
RowGap	Gets or sets the column gap in pixels (px) or percentage (%) for the layout. The row gap is the gap between two successive rows.
Style	Sets the grid cell style. This is a space delimited string that sets for each row (or column) the specifier for each column in the row. The specifier per column (or row) can be based on fractions (fr), pixels (px) or percentage (%). For example, for a grid with 3 equally divided column widths, the style

Property	Description
	could be set to '1fr 1fr 1fr'. For a combination of a fixed column width in pixels of 200 pixels and two columns where the 2nd column has the double width of the third, the style could be set to '200px 2fr 1fr'.
StyleType	Sets the grid cells responsive style to be based on columns (gTemplateColumns) or rows (gTemplateRows).
Tag	Integer property.
Width	Sets the control width in pixels under which the layout is chosen.

TWebMessageDlg

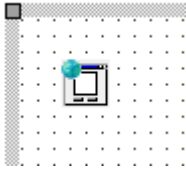
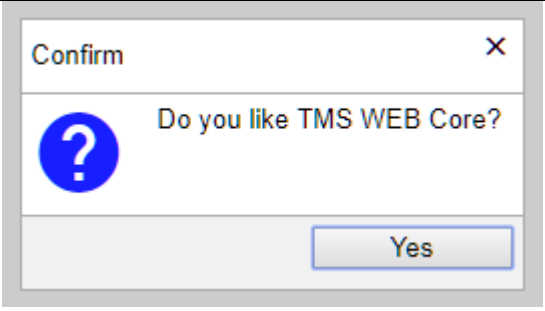


Description

Below is a list of the most important properties methods and events for TWebMessageDlg. This component allows to display modal dialogs (simulated by disabling all controls on the page as the concept of modal dialogs does not exist in web applications).

Result for the following code:

```
WebMessageDlg1.ShowDialog('Do you like TMS WEB  
Core?',WEBLib.Dialogs.mtConfirmation, [mbYes]);
```

 <p>Designtime</p>	 <p>Runtime</p>
---	---

Properties for TWebMessageDlg

DialogResult: TModalResult	Holds the result of calling the dialog
DialogText: TStringList	List of text used in dialog and dialog buttons. Allows for language customization of the dialog text
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file

Methods for TWebMessageDlg

ShowDialog(Msg: string; DlgType: TMsgDlgType; Buttons: TMsgDlgButtons; AProc: TDialogResultProc = nil);	Method to show the message. The last parameter is a method pointer for a method that is optionally called when assigned when the dialog is closed
---	---

Events for TWebMessageDlg

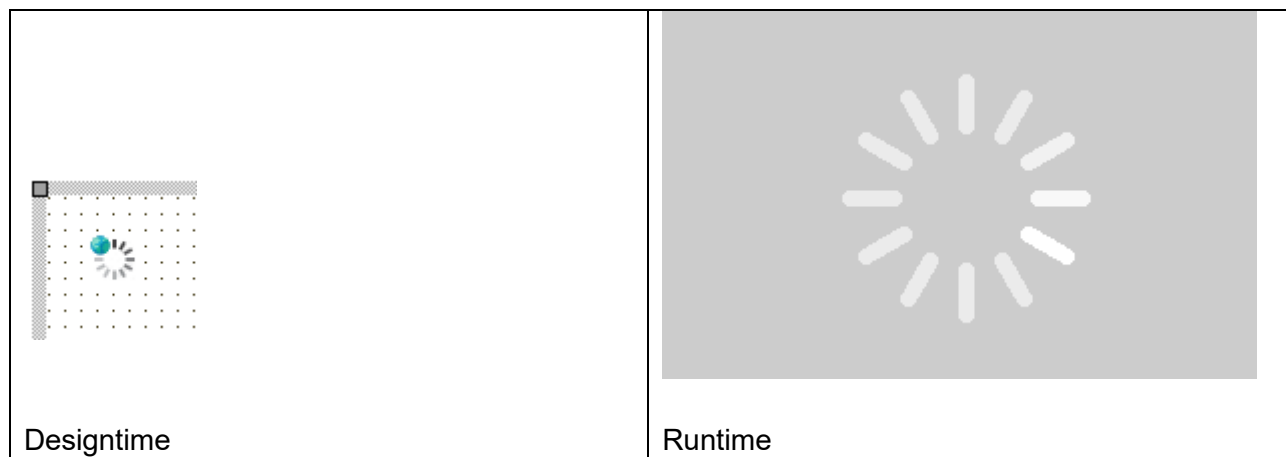
OnButtonClick	Event triggered when a button on the message dialog is clicked
OnClose	Event triggered when the messagebox is closed

TWebWaitMessage



Description

TWebWaitMessage is a non-visual component that enables to show a wait cursor during lengthy operations. TWebWaitMessage shows by default a running wheel animated GIF in the center of the browser window with all controls in the window disabled.



Properties for TWebWaitMessage

Picture: TImage	Image that is displayed while the wait message is active. By default, this is set to an animated GIF with a running wheel. Typically this is an animated GIF.
Opacity: double	Sets the opacity of the layer shown over the window while the wait message is active
Showing: boolean	Returns true while the wait message is being displayed

Methods for TWebWaitMessage

Show	Method to show the wait message.
Hide	Method to hide the wait message

Events for TWebWaitMessage

OnClose	Event triggered when the wait message was closed
---------	--

TWebFileUpload



Description

The file upload component allows the user to drag a local file on the web form or select it via a file open dialog. Either a single file can be uploaded or multiple files.

Properties for TWebFileUpload

Property	Description
Caption	Sets the text to be displayed in the upload component for the button to open the file dialog.
DragCaption	Sets the text to be displayed under the file drag area.
Files	This is a list of files picked. The list consists of objects of the TFile type.
Multifile	When true, it is allowed to pick or drag multiple files.
ShowFiles	When true, the filenames for the dragged or picked files are shown in the control.

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<FORM ID="UniqueID"></FORM>
ElementID	UniqueID

Events for TWebFileUpload

Property	Description
OnDroppedFiles	Event triggered when one or multiple files were dropped or picked. The filenames are returned via the AFileList stringlist.
OnGetFileAsArrayBuffer	Event triggered when the the retrieval or a file as an array buffer is completed. Retrieval of the file is done programmatically by calling WebFileUpload.Files[index].GetFileAsArrayBuffer
OnGetFileAsText	Event triggered when the the retrieval or a file as text is completed. Retrieval of the file is done programmatically by calling WebFileUpload.Files[index].GetFileAsText

Property	Description
OnGetFileAsBase64	Event triggered when the the retrieval or a file as base64 encoded text is completed. Retrieval of the file is done programmatically by calling <code>WebFileUpload.Files[index].GetFileAsBase64</code>
OnGetFileAsDataURL	Event triggered when the the retrieval or a file as Data URL is completed. Retrieval of the file is done programmatically by calling <code>WebFileUpload.Files[index].GetFileAsDataURL</code>
OnUploadFileComplete	Event triggered when an upload of the file is completed. The upload is started with <code>WebFileUpload.Files[index].Upload(AAction);</code>
OnUploadFileResponseComplete	Event triggered when an upload of the file is completed. The upload is started with <code>WebFileUpload.Files[index].Upload(AAction);</code> This event returns the JavaScript request object as well as response as text
OnUploadFileAbort	Event triggered when an upload of the file is aborted
OnUploadFileError	Event triggered when an error has occurred during a file upload
OnUploadFileProgress	Event triggered to indicate the progress of an upload transfer. The event returns the number of bytes transferred from the total number of bytes to transfer

Properties for TFile

TFile is the item in the TWebFileUpload or TWebFilePicker Files collection. After a local file was picked, the Files collection contains the list of files picked and allows access to the file information and file data.

Property	Description
FileObject: TJSHTMLFile	Reference to the HTML TJSHTMLFile object giving accesss to the local file.
Name: string	Returns the name of the local file.
MimeType: string	Returns the MIME type of the local file.
Modified: TDateTime	Returns the file last modified date of the local file.
Size: integer	Returns the size of the local file.
OnGetFileAsText	Event triggered when the retrieval of the local file as text is ready
OnGetFileAsBase64	Event triggered when the retrieval of the local file as base64 encoded data is ready
OnGetFileAsDataURL	Event triggered when the retrieval of the local file as Data URL is ready

Property	Description
OnGetFileAsArrayBuffer	Event triggered when the retrieval of the local file as array buffer is ready

Methods for TFile

Property	Description
GetFileAsText	Starts to retrieve the content of the file as text. When ready the OnGetFileAsText event is triggered at TWebFileUpload or TWebFilePicker level.
GetFileAsText(AEncoding: string)	Overload of GetFileAsText where the text file encoding format can be specified.
GetFileAsText(GetAsString: TGetAsStringProc);	Overload of GetFileAsText that allows the use of an anonymous method to handle the download result.
GetFileAsArrayBuffer	Starts to retrieve the content of the file as binary data (JavaScript array buffer). When ready the OnGetFileAsArrayBuffer event is triggered at TWebFileUpload or TWebFilePicker level.
GetFileAsArrayBuffer (GetAsArrayBuffer: TGetAsArrayBufferProc);	Overload of GetFileAsArrayBuffer that allows the use of an anonymous method to handle the download result.
GetFileAsBase64	Starts to retrieve the content of the file as base64 encode text. When ready the OnGetFileAsBase64 event is triggered at TWebFileUpload or TWebFilePicker level.
GetFileAsBase64(GetAsString: TGetAsStringProc);	Overload of GetFileAsBase64 that allows the use of an anonymous method to handle the download result.
GetFileAsDataURL	Starts to retrieve the content of the file as string that can be used for a data URL for a HTML IMG element
GetFileAsDataURL(GetAsString: TGetAsStringProc);	Overload of GetFileAsDataURL that allows the use of an anonymous method to handle the download result.
Upload(AAction: string);	Perform an upload of a file to a specific upload handler URL AAction
AbortUpload: boolean;	When an upload request is ongoing, it can be aborted by calling AbortUpload. Returns true when this was executed.

Example: uploading a file

To upload a file to a server, from the `WebFileUpload.OnChange` event or from another event, call

```
WebFileUpload1.Files[0].Upload('http://localhost:8088/upload');
```

to upload the first file picked by the `TWebFileUpload` to the server (assuming there is server code listening on port 8088 to handle via the upload action).

TWebFilePicker



Description

The file picker component allows to pick files from the local file system.

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<INPUT TYPE="FILE" ID="UniqueID">
ElementID	UniqueID

Properties for TWebFilePicker

Property	Description
Accept	Sets an optional file filter. This is a string containing the extensions of files that can be selected. Note that setting the file filter will not prevent that the user can pick other filenames. To select only text files (*.txt), set Accept to '.txt'. To select JPEG, GIF, PNG image files , set Accept to '.jpg,.jpeg,.png,.gif' or it could also be set to: 'image/*'.
Files	This is a list of files picked. The list consists of objects of the TFile type.
Multifile	When true, it is allowed to pick or drag multiple files.

Events for TWebFilePicker

Property	Description
OnChange	Event triggered when the file(s) picked changed by the user.
OnGetFileAsArrayBuffer	Event triggered when the the retrieval or a file as an array buffer is completed. Retrieval of the file is done programmatically by calling WebFilePicker.Files[index].GetFileAsArrayBuffer
OnGetFileAsText	Event triggered when the the retrieval or a file as text is

Property	Description
	completed. Retrieval of the file is done programmatically by calling WebFilePicker.Files[index].GetFileAsText
OnGetFileAsBase64	Event triggered when the the retrieval or a file as base64 encode text is completed. Retrieval of the file is done programmatically by calling WebFilePicker.Files[index].GetFileAsBase64
OnGetFileAsDataURL	Event triggered when the the retrieval or a file as Data URL is completed. Retrieval of the file is done programmatically by calling WebFileUpload.Files[index].GetFileAsDataURL
OnUploadFileComplete	Event triggered when an upload of the file is completed. The upload is started with WebFileUpload.Files[index].Upload(AAction);
OnUploadFileResponseComplete	Event triggered when an upload of the file is completed. The upload is started with WebFileUpload.Files[index].Upload(AAction); This event returns the JavaScript request object as well as response as text
OnUploadFileAbort	Event triggered when an upload of the file is aborted
OnUploadFileError	Event triggered when an error has occurred during a file upload
OnUploadFileProgress	Event triggered to indicate the progress of an upload transfer. The event returns the number of bytes transferred from the total number of bytes to transfer

Example code

This code snippet shows how a local file can be loaded in TWebMemo after having been picked by the TWebFilePicker. From the TWebFilePicker.OnChange event, the first picked file is accessed as text with GetFileAsText and from the event TWebFilePicker.OnGetFileAsText this text is added to a TWebMemo.

```

1. procedure TForm2.WebFilePicker1Change(Sender: TObject);
2. begin
3.     if WebFilePicker1.Files.Count > 0 then
4.         WebFilePicker1.Files[0].GetFileAsText;
5. end;
6.
7. procedure TForm2.WebFilePicker1GetFileAsText(Sender: TObject;
8.     AFileIndex: Integer; AText: string);
9. begin
10.    WebMemo1.Lines.Text := AText;
11. end;

```

TWebShare



Description

The TWebShare non-visual component allows to put text, links and/or files on the share sheet of a mobile device from a regular web client application or from a PWA. It is a requirement that the application is hosted on an SSL enabled domain, i.e. accessed via a HTTPS URL.

Methods for TWebShare

Method	Description
Share(ATitle, Atext, AURL: string);	Puts a text on the mobile device share sheet. This can be accompanied by an URL. A title can be set to show in addition to the share dialog on the mobile device.
Share(ATitle, Atext, AURL, AFiles: TJSTHTMLFileArray	Puts a text on the mobile device share sheet. This can be accompanied by an URL. A title can be set to show in addition to the share dialog on the mobile device. In addition to text, an URL, it can also put files on the share sheet. The AFiles parameter is an array of JavaScript file types.
CanShareFiles: boolean	Function returns true when the mobile device browser can also put files on the share sheet

TWebOpenDialog



Description

The TWebOpenDialog non-visual component allows to start a dialog to pick files from the local file system.

Properties for TWebOpenDialog

Property	Description
Accept	Sets an optional file filter. This is a string containing the extensions of files that can be selected. Note that setting the file filter will not prevent that the user can pick other filenames. To select only text files (*.txt), set Accept to '.txt'. To select JPEG, GIF, PNG image files , set Accept to '.jpg,.jpeg,.png,.gif' or it could also be set to: 'image/*'.
FileName	This returns the name of the local file picked
Files	This is a list of files picked. The list consists of objects of the TFile type.
Multifile	When true, it is allowed to pick or drag multiple files.

Methods for TWebOpenDialog

Property	Description
Execute	Starts the dialog for picking a local file
Execute(AProc: TOpenDialogProc);	Starts the dialog for picking a local file with anonymous handler called when a file is selected

Events for TWebOpenDialog


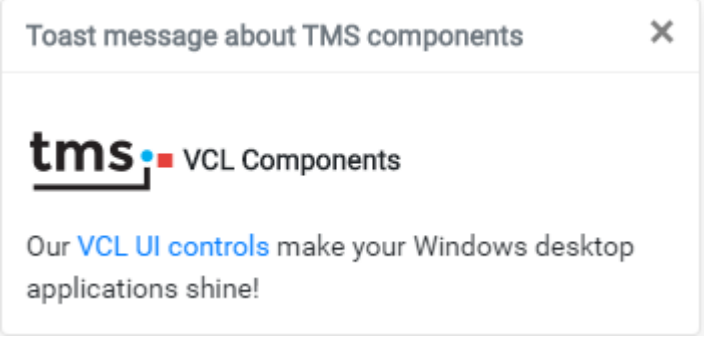
Property	Description
OnChange	Event triggered when the file(s) picked changed by the user.

TWebToast



Description

TWebToast is a non-visual component that enables to show Bootstrap 4.x unobtrusive toast messages on the browser window. Therefore, to use TWebToast, make sure to add the Bootstrap 4.x library and jQuery 3.x library.

 <p>Designtime</p>	 <p>Runtime</p>
--	--

Properties for TWebToast

AutoHideDelay	Sets the time (in milliseconds) for a toast message to automatically hide (when enabled)
Container	Sets an optional container control that is used to control the position where the toast message will display
Items	Collection of toast message items of type TToastItem
Position	Sets the position where the toast messages will appear on the screen tpAbsolute : uses the X,Y properties to set the absolute position

	<p>tpTopLeft : toast messages appear in the top left corner</p> <p>tpTopRight : toast messages appear in the top right corner</p> <p>tpBottomLeft : toast messages appear in the bottom left corner</p> <p>tpBottomRight : toast messages appear in the bottom right corner</p> <p>tpContainer : toast messages appear within the specified container control</p>
--	---

Events for TWebToast

OnHide	Event triggered when the toast message hides.
--------	---

Properties for TToastItem

AutoHide	When true, the item will automatically hide after a delay set via TWebToast.AutoHideDelay
Body	Sets the body text for the toast item
CloseButton	When true, a close button will appear in the top right corner of the toast message
Header	Sets the header text for the toast item
Time	<p>Sets the type of the time displayed in the toast message:</p> <p>ttNone: no time is displayed</p> <p>ttShow: shows the absolute time when the toast message is displayed</p> <p>ttDeltaShow: shows the time difference between the current time and the time at which the toast message was displayed</p>

Methods for TToastItem

Show	Shows the toast message on the screen
Hide	Hides the toast message from the screen

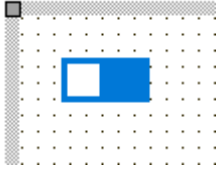

Update	When properties change for an existing TWebToastItem that is already displayed, call WebToastItem.Update
--------	--

TWebToggleButton



Description

Below is a list of the most important properties methods and events for TWebToggleButton.

 <p>Designtime</p>	 <p>Runtime</p>
--	---

Properties for TWebToggleButton

Checked	Sets or gets the state of the toggle button
Style	Style of the toggle button can be tsRectangular or tsRounded

Events for TWebToggleButton

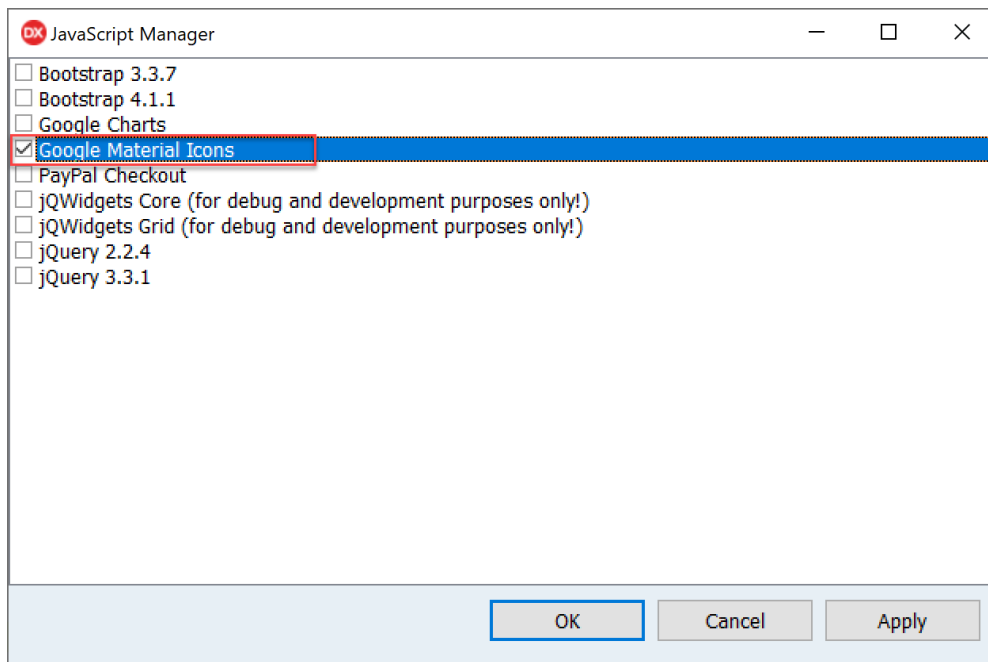
OnClick	Event triggered the toggle button is clicked
---------	--

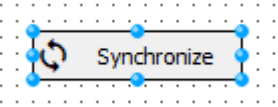
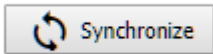
TWebBitBtn



Description

Below is a list of the most important properties methods and events for TWebBitBtn.
Note that TWebBitBtn uses the Google Material Icons. Make sure to include this library in your project. (Select the Manage JavaScript Library from the project context menu)



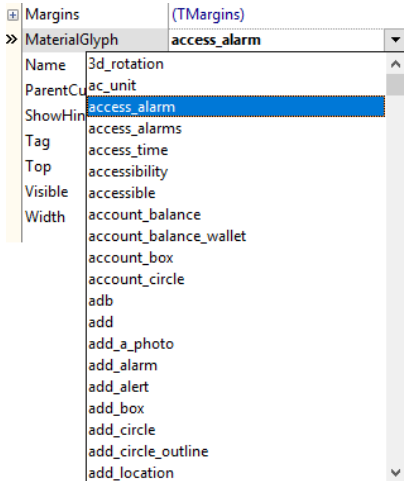
 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<BUTTON ID="UniqueID"></BUTTON>
ElementID	UniqueID

Properties for TWebBitBtn

Caption	Sets the caption for the button
Flat	When true, the button is displayed in flat style
Glyph	Sets the optional image for the button
Layout	Sets the position of the button image versus the button caption blGlyphLeft: glyph left from caption blGlyphRight: glyph right from caption blGlyphTop: glyph on top of caption blGlyphBottom: glyph under caption
MaterialGlyph	Allows to pick an icon from the Google material icon set 
MaterialGlyphColor	Sets the color of the material glyph icon
MaterialGlyphSize	Sets the size of the material glyph

Events for TWebBitBtn

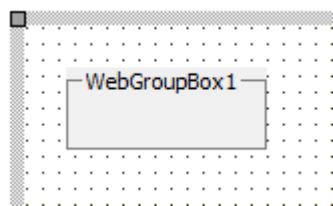
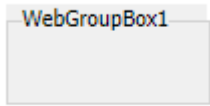
OnClick	Event triggered when the button is clicked
---------	--

TWebGroupBox



Description

Below is a list of the most important properties methods and events for TWebGroupBox. The TWebGroupBox is a container control with a caption

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	
ElementID	UniqueID

Properties for TWebGroupBox

ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file

Events for TWebGroupBox

OnClick	Event triggered when the groupbox is clicked
OnDbClick	Event triggered when the groupbox is double-clicked

TWebStretchPanel



Description

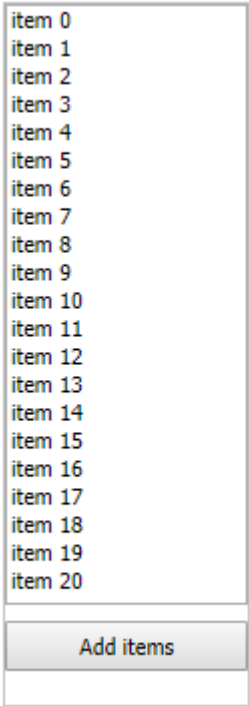
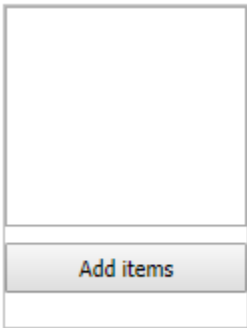
Below is a list of the most important properties methods and events for TWebStretchPanel. The TWebStretchPanel is a container control with a top and bottom area. The height of the bottom area has a fixed height while the top area height can adapt itself to the height of controls (when controls are relatively positioned in the top area).

When a control is put in the upper area at design-time, it will belong at runtime in the upper stretching area of the TWebStretchPanel. When a control is put in the lower area, it will belong to the lower fixed height area and will as such automatically appear lower when the upper panel area is stretched to fit the controls in the upper area.



Example:

A TWebListBox and TWebButton is placed on the TWebStretchPanel. The button is on the lower part, the listbox on the upper part. From the button, items are added to the listbox and the height of the listbox is increased. This causes the upper part to stretch to the height of the listbox and the button remains below the stretched upper area in the fixed height area of the lower part:



```
procedure TSampleForm.WebButton1Click(Sender: TObject);
var
  i: integer;
begin
  for i := 0 to 20 do
  begin
    WebListbox1.Items.Add('item '+inttostr(i));
  end;
  WebListbox1.Height := 300;
end;
```

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebStretchPanel

ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
FixedHeight	Sets the fixed height of the bottom area in the panel.

Events for TWebStretchPanel

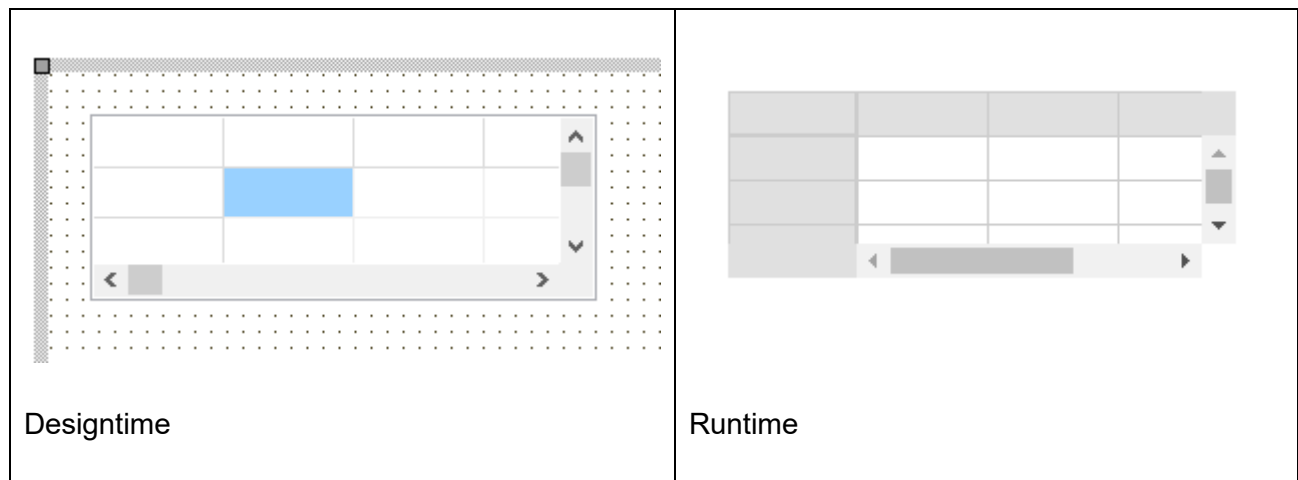
OnClick	Event triggered when the groupbox is clicked
OnDbClick	Event triggered when the groupbox is double-clicked
OnMouseDown	Event triggered when the mouse is down on the panel
OnMouseMove	Event triggered when the mouse moves over the panel
OnMouseUp	Event triggered when the mouse goes up on the panel

TWebStringGrid



Description

Below is a list of the most important properties methods and events for TWebStringGrid. TWebStringGrid is similar to a VCL TStringGrid.



Set or get the content of grid cells via:

Grid.Cells[col,row]: string;

Set or get the column width in the grid via

Grid.ColWidths[col]: integer;

Set or get the row height in the grid via

Grid.RowHeights[row]: integer;

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<TABLE></TABLE>
ElementID	UniqueID

Properties for TWebStringGrid

BorderStyle	Selects the border style of the grid
ColCount	Sets the number of columns in the grid
DefaultColWidth	Sets the default column width
DefaultRowHeight	Sets the default row height
FixedColor	Sets the color of fixed cells
FixedCols	Sets the number of fixed columns in the grid
FixedRows	Sets the number of fixed rows in the grid
LeftCol	Gets or sets the index of the first normal grid column displaying. Use this property to get or set the horizontal scroll position
Options	The settings that are supported are: goEditing: enables editing in the grid goHorzLine: enables horizontal grid lines goVertLine: enables vertical grid lines goRowSelect: enables row selection
RowCount	Sets the number of rows in the grid
Selection	Gets or selects the range of selected cells in the grid. Selection is of the type TGridRect
TopRow	Gets or sets the index of the first normal grid row displaying. Use this property to get or set the vertical scroll position

Methods for TWebStringGrid

InsertRow(Index: integer);	Inserts a new row in the grid at Index
LoadFromJSON(const AURL: string; ADataNode: string);	Load JSON formatted data found a AURL via a HTTP GET in the string grid. The expected data is a JSON array. When the ADataNode parameter is different from empty, it tries to fetch the JSON array from the ADataNode JSON node.

LoadFromJSON(AJSON: TJObject; ADataNode: string);	Load data from a JSON object. The expected data is a JSON array. When the ADataNode parameter is different from empty, it tries to fetch the JSON array from the ADataNode JSON node.
LoadFromCSV(const AURL: string; Delimiter: char = ';'; LoadFixed: Boolean = false)	Load CSV formatted data found a AURL via a HTTP GET in the string grid. Optional parameters are the delimiter to use to parse the CSV file and when the LoadFixed parameter is true, the CSV data is also loaded in the fixed cells of the grid.
MouseToCell(X,Y: integer; var ACol,ARow: integer);	Returns the column/row index of the cell found at client coordinates X,Y of the grid
RemoveRow(Index: integer);	Removes row Index from the grid

Events for TWebStringGrid

OnCanEditCell	Event triggered just before editing starts (when goEditing = true in grid.Options) with a var parameter CanEdit to control whether the cell can be edited or not
OnClick	Event triggered when grid is clicked
OnDbClick	Event triggered when grid is double-clicked
OnGetCellChildren	Event triggered when a new cell is rendered during loading data from CSV or JSON in the grid. Passes the HTML element for the grid cell allowing to insert dynamically HTML child elements in the cell
OnGetCellClass	Event triggered when a new cell is rendered during loading data from CSV or JSON in the grid. Allows to set the CSS class name for an individual cell allowing customization this way.
OnGetCellData	Event triggered when a new cell is rendered during loading data from CSV or JSON in the grid. Allows to dynamically override or customize the values retrieved from the CSV or JSON (or dataset in case of a

	TWebDBGrid)
OnGetEditText	Event triggered when a cell goes to edit mode requesting the value to be edited
OnHttpRequestError	Event triggered when an error occurred with the HTTP GET request used to get data via methods LoadFromJSON()/LoadFromCSV()
OnHttpRequestSuccess	Event triggered when the HTTP GET request used to get data via methods LoadFromJSON()/LoadFromCSV() successfully returned
OnSetEditText	Event triggered when a cell goes out of edit mode returned the edited value

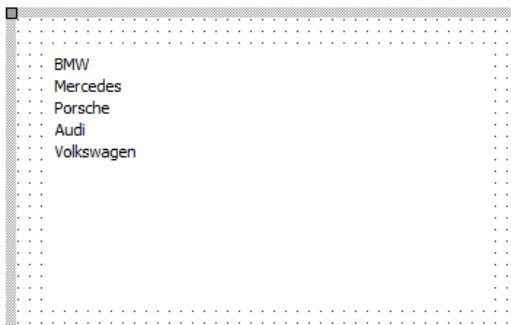
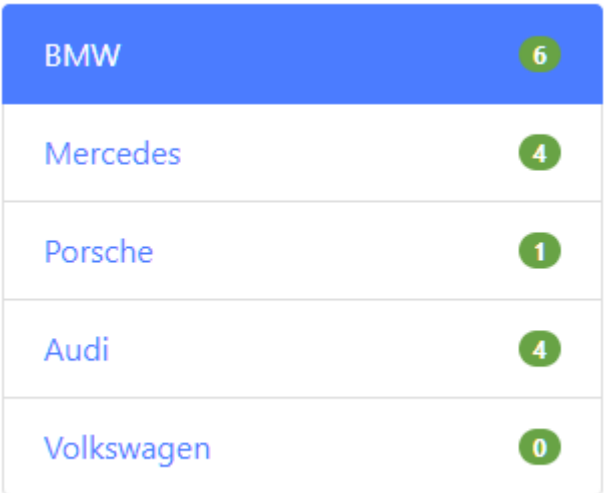
TWebListControl



Description

Below is a list of the most important properties methods and events for TWebListControl. TWebListControl represents a HTML list structure. The TWebListControl is also especially designed to be able to use Bootstrap CSS styles for effects like banding, hovering,... and much more. Find more information about Bootstrap list styles at: <https://getbootstrap.com/docs/4.0/components/list-group>

In this example, the ElementListClassName was set to: "list-group" and the item's property ItemClassName was set to: "list-group-item d-flex justify-content-between align-items-center list-group-item-action"

 <p>Designtime</p>	 <p>Runtime</p>
---	---

Items are added to the list via the Items collection. The Item class is defined as:

Properties for TListItem

Active	When true, the item is shown as active item in the list (when the CSS defines the Active style)
AutoCollaps	When true, the item click will collapse / uncollapse the sub items (when the CSS defines the Collapse style)
Enabled	When true, the item is enabled and can be clicked and will trigger the OnItemClick event
ItemClassName	Optionally sets the CSS classname for the item when styling via CSS is used
Items	Collection of sub items for an item. The sub items collection is exactly the same as the main items collection. Note that items in sub items can also have sub items etc..
Link	Sets the optional URL for the item text when it needs to be clickable with an URL reference
LinkClassName	Optionally sets the CSS classname for the item link when styling via CSS is used
Tag	Integer tag property associated with the item
Text	Text of the item

Methods for TListItem

Expand	When the item has subitems, expands the subitems
Collapse	When the item has subitems, collapses the subitems
IsCollapsed	When true, the subitems of the item are in collapsed state

Properties for TWebListControl

DefaultItemClassName	Sets the CSS class that is automatically applied to an item ItemClassName when a
----------------------	--

	new item is created. The DefaultItemClassName is only used upon creation of new TLinkItem instances
DefaultItemLinkClassName	Sets the CSS class that is automatically applied to an item LinkClassName when a new item is created. The DefaultLinkClassName is only used upon creation of new TLinkItem instances
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
ElementListClassName	Optionally sets the CSS classname for the list when styling via CSS is used
Items	Collection of TListItem instances and possibly sub items making up the list
Style	<p>When Style is set, this presets the CSS DefaultItemClassName, DefaultItemLinkClassName, ElementListClassName to match popular Bootstrap list styles.</p> <p>Sets the style of the list to:</p> <p>IsBreadCrumb: list of items makes up a breadcrumb</p> <p>IsListGroup: vertical list of items</p> <p>IsPagination: list makes up items of a paging control, like a control to select a page of rows to show in a grid</p> <p>IsTabs: list makes up items of tab group</p>

Events for TWebListControl

OnGetItemChildren	Event triggered when the list item is rendered allowing to insert child HTML elements in the list element
-------------------	---

OnGetItemClass	Event triggered when the list item is rendered allowing to customize the CSS class of the list element
OnItemClick	Event triggered when a list item is clicked
OnItemDbClick	Event triggered when a list item is double-clicked

TWebTableControl



Description

Below is a list of the most important properties methods and events for TWebTableControl. TWebTableControl represents a HTML table. The HTML table can have a header row and/or header column. The TWebTableControl is also especially designed to be able to use Bootstrap CSS styles for effects like banding, hovering,... Find more information about Bootstrap table styles at: <https://getbootstrap.com/docs/4.0/content/tables/>

In this example, the ElementHeaderClassName was set to: "table thead-dark" and the ElementTableClassName was set to: "table table-hover table-bordered table-striped table-sm"

<p>Designtime</p>	<p>Runtime</p>
-------------------	----------------

Set or get the content of table cells via:

TableControl.Cells[col,row]: string;

Set or get the HTML table cell elements in the grid via:

TableControl.CellElements[col,row]: TJSElement

Set or get the CSS class name for a row in the table via:

TableControl.RowClassName[row]: string;

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	
ElementID	UniqueID

Properties for TWebTableControl

ColCount	Sets the number of columns in the table
ColHeader	When true, a row header column is shown
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementHeaderClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
ElementTableClassName	Optionally sets the CSS classname for the label when styling via CSS is used
RowCount	Sets the number of rows in the table
RowHeader	When true, a column header row is shown

Methods for TWebTableControl

LoadFromJSON(const AURL: string; ADataNode: string);	Load JSON formatted data found a AURL via a HTTP GET in the string grid. The expected data is a JSON array. When the ADataNode parameter is different from empty, it tries to fetch the JSON array from the ADataNode JSON node.
LoadFromCSV(const AURL: string; Delimiter: char = ','; LoadFixed: Boolean = false)	Load CSV formatted data found a AURL via a HTTP GET in the table contro. Optional parameters are the delimiter to use to parse the CSV file and when the LoadFixed parameter is true, the CSV data is also loaded in the fixed cells of the table control.

Events for TWebTableControl

OnClick	Event triggered when the table is clicked
OnClickCell	Event triggered when a table cell is clicked
OnDbClick	Event triggered when the table is double-clicked
OnDbClickCell	Event triggered when a table cell is double-clicked
OnGetCellChildren	Event triggered when a new cell is rendered during loading date from CSV or JSON in the grid. Passes the HTML element for the grid cell allowing to insert dynamically HTML child elements in the cell
OnGetCellClass	Event triggered when a new cell is rendered during loading date from CSV or JSON in the grid. Allows to set the CSS class name for an individual cell allowing customization this way.
OnGetCellData	Event triggered when a new cell is rendered during loading date from CSV or JSON in the grid. Allows to dynamically override or customize the values retrieved from the CSV or JSON (or dataset in case of a TWebDBGrid)

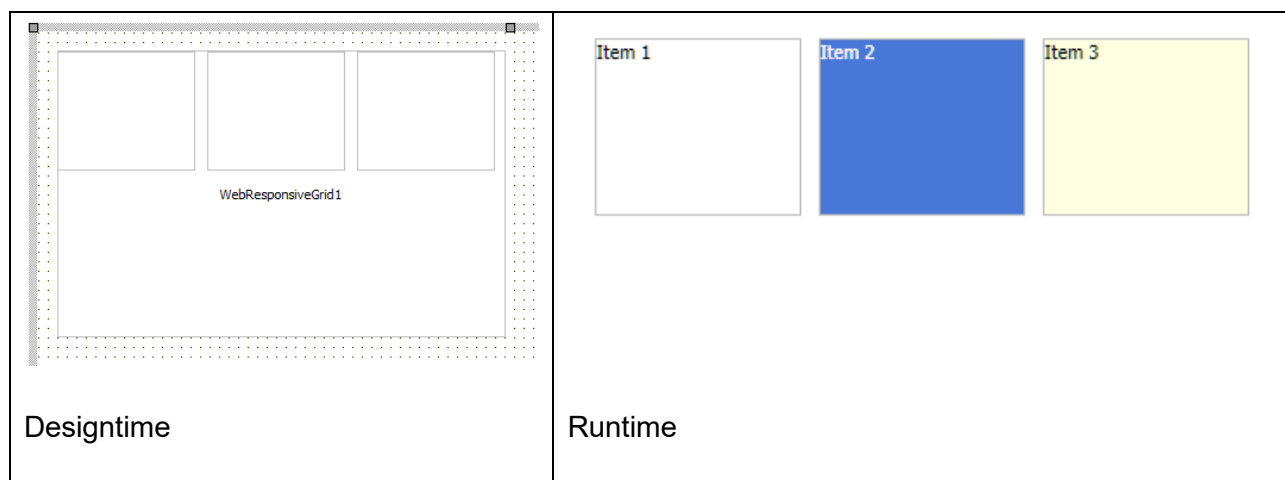
OnHttpRequestError	Event triggered when an error occurred with the HTTP GET request used to get data via methods LoadFromJSON()/LoadFromCSV()
OnHttpRequestSuccess	Event triggered when the HTTP GET request used to get data via methods LoadFromJSON()/LoadFromCSV() successfully returned

TWebResponsiveGrid



Description

Below is a list of the most important properties methods and events for TWebResponsiveGrid. TWebResponsiveGrid represents a HTML table structure with a responsive behavior of configuration of columns and rows in relationship to the screen size the control is rendered on.



The TWebResponsiveGrid renders items from its Items collection in columns and rows. The number of columns and rows can dynamically adapt to the size of the screen on which the control is rendered.

To add items to TWebResponsiveGrid, use the Items collection and set the HTML content for each item via `WebResponsiveGrid.Items[index].HTML: string;`

For each item, there is also a `Tag: integer` property and `ItemObject: TObject` property for setting information associated with the item.

The HTML element in the grid via which the item is rendered is also accessible via public property `WebResponsiveGrid.Items[index].ElementHandle: TJSHTMLInputElement.`

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebResponsiveGrid

Configuration of the responsive behavior of the control is set via the Options property.

ItemBorderColor	Sets the border color of an item in normal state
ItemClassName	Sets the CSS class name for an item
ItemColor	Sets the background color of an item in normal state
ItemGap	Sets the gap (horizontally and vertically) in pixels between items in the grid
ItemHeight	Sets the height of an item in pixels
ItemHoverBorderColor	Sets the border color of an item in hovered state
ItemHoverColor	Sets the background color of an item in hovered state
ItemPadding	Sets the padding internally in an item in pixels
ItemSelectedBorderColor	Sets the border color of an item in selected state
ItemSelectedColor	Sets the background color of an item in selected state
ItemSelectedTextColor	Sets the text color of an item in selected state
ItemTemplate	Sets an optional HTML template to be used when data for the responsive grid is dynamically loaded from CSV or JSON. Use (%FIELDNAME%) place-holders in the HTML template to define which data should be used in what parts of the HTML for the item. In addition, the placeholder (%ITEMINDEX%) can be used to generate

	the index of the item in the item collection in the resulting HTML.
ItemMinWidth	Sets the minimum width of an item in pixels. This will determine the number of columns that can be rendered in the grid.
ScrollVertical	When true, a vertical scrollbar will be used when the number of items exceeds the height of the control. Otherwise, the height will automatically increase to enable to display of all items in the list.

Methods for TWebResponsiveGrid

LoadFromJSON(const AURL: string; ADataNode: string);	Load JSON formatted data found a AURL via a HTTP GET in the string grid. The expected data is a JSON array. When the ADataNode parameter is different from empty, it tries to fetch the JSON array from the ADataNode JSON node.
LoadFromCSV(const AURL: string; Delimiter: char = ';'; LoadFixed: Boolean = false)	Load CSV formatted data found a AURL via a HTTP GET in the table contro. Optional parameters are the delimiter to use to parse the CSV file and when the LoadFixed parameter is true, the CSV data is also loaded in the fixed cells of the table control.
ItemByTag(ATag: integer): TResponsiveGridItem	Returns the item with the specified tag value when it exists or nil when not
ListElementHandle: TJSElement	Returns the DIV HTML element that is the container element of the responsive grid HTML elements

Events for TWebResponsiveGrid

OnClick	Event triggered when the grid is clicked
OnDbClick	Event triggered when the grid is double-clicked
OnHttpRequestError	Event triggered when there is a HTTP error related to loading data from CSV or JSON.

OnHttpRequestSuccess	Event triggered when the HTTP get request to get data from CSV or JSON was successful.
OnGetCellChildren	Event triggered when a new cell is rendered during loading data from CSV or JSON in the grid. Passes the HTML element for the grid cell allowing to insert dynamically HTML child elements in the cell
OnGetCellClass	Event triggered when a new cell is rendered during loading data from CSV or JSON in the grid. Allows to set the CSS class name for an individual cell allowing customization this way.
OnGetCellData	Event triggered when a new cell is rendered during loading data from CSV or JSON in the grid. Allows to dynamically override or customize the values retrieved from the CSV or JSON (or dataset in case of a TWebDBGrid)
OnHttpRequestError	Event triggered when an error occurred with the HTTP GET request used to get data via methods LoadFromJSON()/LoadFromCSV()
OnHttpRequestSuccess	Event triggered when the HTTP GET request used to get data via methods LoadFromJSON()/LoadFromCSV() successfully returned
OnItemClick	Event triggered when an item in the grid is clicked
OnItemCreated	Event triggered when an item in the grid is created as a result of loading data from a CSV file or JSON file. The Item can be accessed via WebResponsiveGrid.Items[index] and the HTML element in which the item is rendered via WebResponsiveGrid.Items[index].ElementHandle: TJSHTMLElement
OnItemDbClick	Event triggered when an item in the grid is double-clicked
OnItemGetFieldValue	Event triggered when a value from a CSV column or JSON field is going to be replaced in the HTML template and via this event, the data can be dynamically customized.

Properties for TWebResponsiveGridItem

Public properties

ElementHandle: TJSHTMLElement	Access to the HTML DIV container element of the item
JSONElement: JSValue	JSON object associated with the item in case items were loaded from a JSON array
JSONElementValue['name']: string	Gets the JSON object 'name' attribute value as string

Published properties

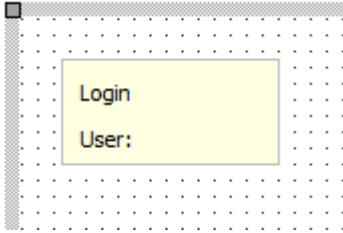
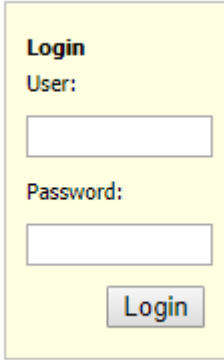
HTML	HTML content of the item
Tag	Integer property
Visible	Sets whether the item is visible or not in the responsive grid

TWebLoginPanel



Description

Below is a list of the most important properties methods and events for TWebLoginPanel.

 <p>Designtime</p>	 <p>Runtime</p>
--	--

Properties for TWebLoginPanel

ElementButtonClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementCaptionClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementInputClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementLabelClassName	Optionally sets the CSS classname for the label when styling via CSS is used
Padding	Sets the padding inside the login panel
Password	Gets or sets the value of the password INPUT control

PasswordLabel	Sets the caption text for the label in front of the password INPUT control
User	Gets or sets the value of the username INPUT control
Userlabel	Sets the caption text for the label in front of the username INPUT control

Events for TWebLoginPanel

OnLogin	Event triggered when the login button is clicked
---------	--

TWebImageSlider



Description

In many scenarios, people want to show various pictures of things for specific items. Think about a product on Amazon that might have different pictures taken from different angles, think about an online real-estate broker presenting different houses with picture sets of the house on sale or a car dealer showing cars for sale accompanied by pictures of the car in various positions.

If you have such a use-case in your application, TWebImageSlider is the shortcut to achieve this. Basically this is a container control where you add the links to the images to be displayed and the control does everything else. It shows the picture thumbnails, a left / right slider button and you can click on thumbnails to see the large version of a specific picture.



Properties for TWebImageSlider

Appearance.Bullets.Color	Sets the color of the bullet indicating the
--------------------------	---

	inactive image
Appearance.Bullets.ColorActive	Sets the color of the bullet indicating the active image
Appearance.Bullets.Opacity	Sets the opacity (between 0 and 1) of the bullets to perform the navigation between images in the slider
Appearance.Bullets.Size	Sets the size (in pixels) of the bullets
Appearance.Bullets.SpaceBetween	Sets the space (in pixels) between bullets
Appearance.Bullets.SpaceEdge	Sets the rounding (in pixels) of the bullets for thumbnail navigation
Appearance.Buttons.Color	Sets the color of the next / preview arrows
Appearance.Buttons.Visible	When true, the navigation next / navigate previous button is visible
Appearance.NavigationStyle	Sets the type of navigation in the image slider as bullets, thumbnails or none
Appearance.Thumbnails.ColorActiveBorder	Sets the border color around the active thumbnail item
Appearance.Thumbnails.NumDisplayed	Sets how many thumbnail images are displayed under the active image
Appearance.Thumbnails.Opacity	Sets the opacity of the thumbnail items
Appearance.Thumbnails.OpacityActive	Sets the opacity of the active thumbnail item
Appearance.Thumbnails.SizePercent	Sets how much % of the original image size the thumbnails have
Appearance.Thumbnails.SpaceBetween	Sets the horizontal space between thumbnails in pixels
Appearance.Thumbnails.WidthActiveBorder	Sets the border width around the active thumbnail in the list
ImageURLs: TStringList	String list holding the URLs for all images in the TWebImageSlider

Public properties for TWebImageSlider

ActiveImageIndex	Index of the selected (active) image in the TWebImageSlider
PreviousActiveImageIndex	Index of the previously selected image
LastClickedImageIndex	Index of the image clicked

Methods for TWebImageSlider

RefreshImages	Call when one or more images in the ImageURLs string list was changed
---------------	---

Events for TWebImageSlider

OnImageChange	Event triggered when the selected (active) image is changed
---------------	---

Example code

This code snippet shows how to load new images in the TWebImageSlider and display tese:

```
var
  i: Integer;
begin
  for i := 1 to 8 do
    ImageSlider.ImageURLs.add(Format('./images/nature-%d.jpg', [i]));
  ImageSlider.RefreshImages;
end;
```

TWebContinuousScroll



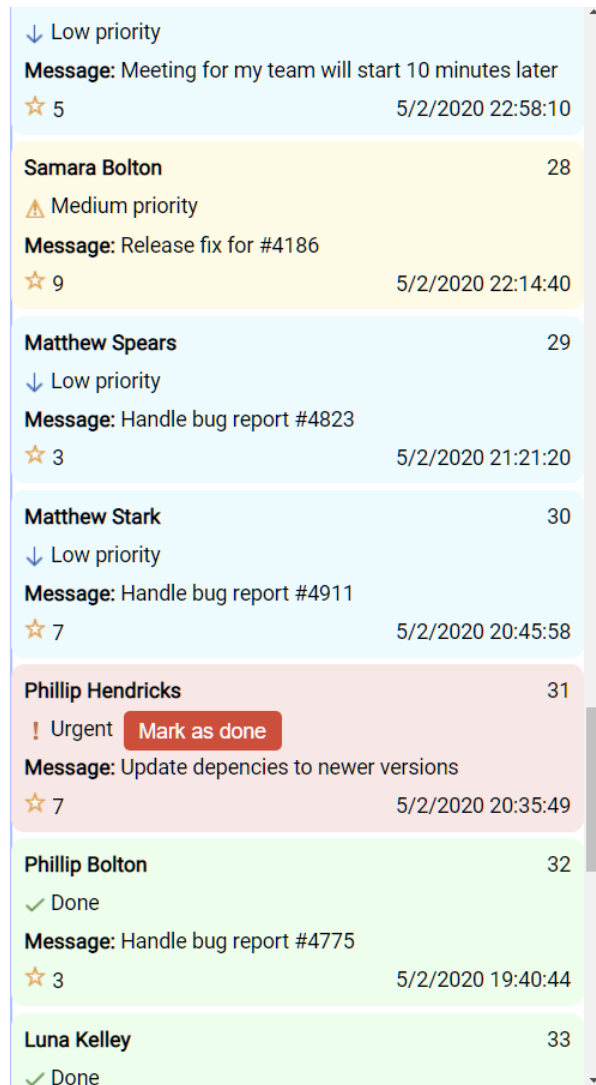
Description

The TWebContinuousScroll control offers the often used functionality in modern web client applications to show lists of items filling the viewing area of the browser only and after this, only load additional items when the user decides to scroll down. The reasoning behind such UI control is simple. By loading only the items in view, the initial display of the page is very fast and only when the user wants to see additional items, extra items are loaded asynchronously in the list.

The TWebContinuousScroll works by requesting page per page of items for the list from the server. The server is expected to return the items as an array of JSON objects. Each JSON object is then rendered as an item in the list.

The request URL per page is set via an event. The component will perform the HTTP request and will then trigger an event for each JSON object in the array to render it as an item.

Additional events are offered in case the server responds in a different way than returning an array of JSON objects.



Properties for TWebContinuousScroll

ButtonText	When the LoadType is ItButton, then a button is displayed at the bottom of the list from where a click will load extra items. ButtonText sets the caption of this button
ElementButtonClassName	Optionally sets the CSS classname for the load button when styling via CSS is used
ElementListEndClassName	Optionally sets the CSS classname for the information displayed at the end of the list when no more items can be loaded when styling via CSS is used
ElementLoadClassName	Optionally sets the CSS classname for the label when styling

	via CSS is used
ItemTemplate	Sets the HTML template for filling the item content when it gets loaded. Data placeholders are added in the HTML template as (%placeholdername%)
ListEndText	Sets the text displayed when the last item is retrieved indicating to the users no more items are on the server
LoadScrollPercent	Sets the value in percent of the scroll range from where automatic loading of extra items should happen when the user is scrolling (when LoadType is set to ItScroll)
LoadType	Sets the way extra items are loaded: ItButton: extra items are loaded when the button at the bottom of the list is clicked ItScroll: extra items are loaded when the user scrolled beyond LoadScrollPercent of the scroll range ItNone: no built-in loading of extra items is happening, extra items are only loaded programmatically
PageNumber	Sets the page number for items to load
PageSize	Sets the number of items per page to load
PostData	Sets the data that is posted along with the page number and item count when a next page of data is requested in mode rmPOST
RequestMode	Sets the HTTP request type to use for fetching a next page of items for the list. This can be a HTTP(s) GET request (rmGET) or a HTTP(s) POST request (rmPOST)
ShowEnd	When true, it is indicated that the end of the list of items is reached
ShowLoading	When true, a progress indicator is shown during the loading of extra items

Methods for TWebContinuousScroll

FetchNextPage	Method will load the next page of extra items in the list
---------------	---

Events for TWebContinuousScroll

OnFetchNextPage	Event triggered when a new page of items needs to be fetched from the server. The URL for the fetch is expected to be returned via the parameter AURL of the event. The event also returns the index of page for which to request items as well as the page
-----------------	--

	size. This should be sufficient to create the URL for most servers to fetch the next list of items.
OnGetData	This event is triggered when the AURL string parameter from OnFetchNextPage remains empty. A TJSArray can be passed as an object array that contains the data to be displayed. If there's no more data to be displayed, set the ALoadMoreData parameter to False.
OnGetListItem	This event is triggered for each JSON object returned from the server after requesting a new page. It enables to dynamically render the content per item. The event returns the index of the item, the JSON object for the item and the HTML container element for the item in the list. This way, code can be added to the event handler to configure the HTML element childs for the item. Note that when the ItemTemplate contains a data container placeholder identification, i.e. (%placeholdername%), this placeholder data will be set to the value found in the JSON object having the attribute name equal to this placeholder name.
OnGetListItemFieldValue	This event is triggered for each placeholder ID found in the ItemTemplate. This allows not only to transform the value to a display value for the item in the list but also to add placeholders that are dynamically mapped to other values. The placeholder name is returned as AFieldName parameter and the value that will be set as placeholder data is expected to be returned via the AValue var parameter.
OnJSONToItem	This event permits to provide the custom conversion of a JSON object to the HTML to be used for the item in the list. Return this HTML via the var parameter AHTML for the AObject parameter TJSONObject.
OnObjectToArray	In case the JSON returned by the server is not a JSON array but maybe a JSON object with a node containing the array, this event can be used to return the proper node from the returned JSON from the server. The parameter AObject.jsobject is the JSON object returned from the server and the event handler should return the JSON array from this object via the AArray parameter.
OnPageLoaded	This event is triggered when the page has been rendered after data was retrieved from the server.
OnResponseToArray	This event is triggered returning the raw text data for the server response. In case this data is not formatted as JSON

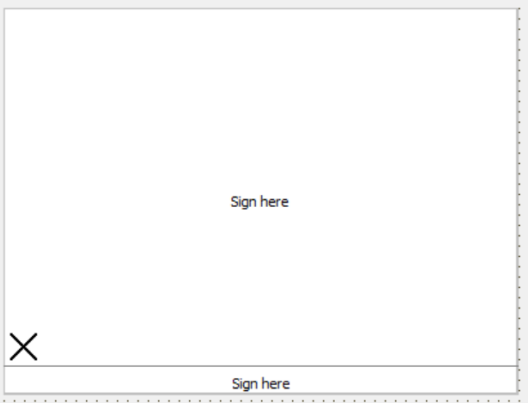

	data, it permits to parse the data and return it as a JSON array to the control for rendering.
--	--

TWebSignatureCapture



Description

Below is a list of the most important properties and methods for TWebSignatureCapture. This component allows capturing a signature from the user in an application.

 <p>Design-time</p>	 <p>Runtime</p>
---	--

Properties for TWebSignatureCapture

ClearButton	Various settings for the clear button.
Empty	Public property that returns if the canvas is empty.
Pen	Settings for the pen.
TextPosition	Various settings for the text.
Text	Optional text to be shown. Default is "Sign here".

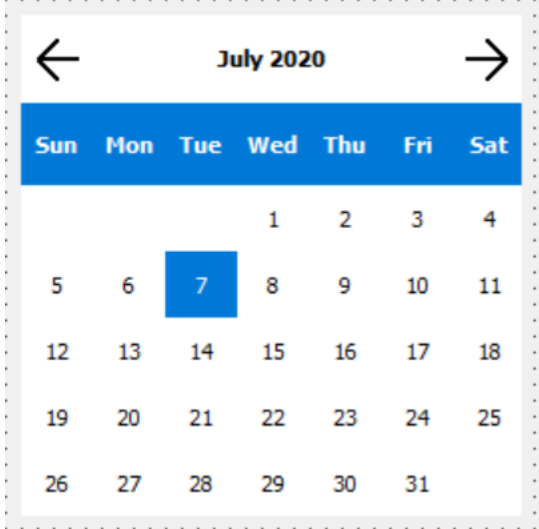
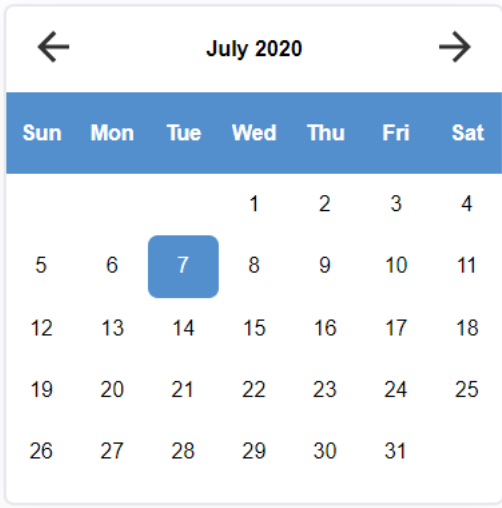
Methods for TWebSignatureCapture

GetAsBase64Image	Returns the signature as a base64 encoded image.
------------------	--

TWebCalendar



Below is a list of the most important properties, methods and events for TWebCalendar.

 <p>Designtime</p>	 <p>Runtime</p>
--	--

Properties for TWebCalendar

Day	Sets/gets the selected day.
ElementBackgroundClassName	Optionally sets the CSS classname for the background
ElementCurrentDateClassName	Optionally sets the CSS classname for the current date
ElementDayNamesClassName	Optionally sets the CSS classname for the day names
ElementHeaderClassName	Optionally sets the CSS classname for the header
ElementSelectedDateClassName	Optionally sets the CSS classname for the selected date
EnablePastDates	Disable or enable the selection of past

	dates.
FirstDay	Sets the first day of the week.
HintNext	Sets a hint for the next button.
HintPrev	Sets a hint for the previous button.
InactiveDays	Sets the inactive days (for example: InactiveDays.Monday := True sets all mondays as inactive).
MaxDate	Sets the maximum date.
MinDate	Sets the minimum date.
Month	Sets/gets the selected month.
MultiSelect	Enable/disable selection of multiple dates.
NameOfDays	Change the displayed names of the days.
NameOfMonths	Change the displayed names of the months.
SelectedDate	Sets/gets the selected date.
ShowToday	If enabled then today's date is highlighted.
Year	Sets/gets the selected year.

Methods for TWebCalendar

SelectedDates	Returns a set of selected dates.
---------------	----------------------------------

Events for TWebCalendar

OnDateSelected	Event triggered when a date is selected.
OnDateUnselected	Event triggered when a date is unselected.

TWebGoogleReCaptcha



Description

The TWebGoogleReCaptcha implements v3 of the Google ReCaptcha API. Below is a list of the most important properties methods and events for TWebGoogleReCaptcha.

 <p>Designtime</p>	 <p>Runtime</p>
---	--

Properties for TWebGoogleReCaptcha

APIKey	A valid Google API Key is required
APIUrl	Sets the URLfor the backend API

Methods for TWebGoogleReCaptcha

Verify(Action)	Start the ReCaptcha verification process. An optional Action string value can be provided which is returned with the OnVerified event.
----------------	--

Events for TWebGoogleReCaptcha






OnVerified	Event triggered when the ReCaptcha verification process has finished. Returns the verification results in the Args parameter values: Action, Score, TimeStamp, HostName
------------	---

TWebGoogleDrive



Description

Below is a list of the most important properties methods and events for TWebGoogleDrive.

<div data-bbox="194 772 824 1226">  </div> <p>Designtime</p>	<p>TITLE</p> <hr/> <div data-bbox="894 787 1127 863">  document.doc 2:38 am </div> <hr/> <div data-bbox="894 911 1070 987">  intro.mp4 2:39 am </div> <hr/> <div data-bbox="894 1035 1088 1110">  sample.jpg 1/26/15 </div> <hr/> <div data-bbox="894 1159 1076 1234">  Winter.jpg 2:38 am </div> <hr/> <p>Runtime</p>
--	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<IFRAME ID="UniqueID"></IFRAME>
ElementID	UniqueID

Properties for TWebGoogleDrive



ElementClassName	Optionally sets the CSS classname for the map when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the component needs to be connected with. When connected, no new object is created but the Delphi class is connected with the existing HTML element in the form HTML file
FolderID	Sets the ID of the Google Drive Folder to display
View	Sets if the files are displayed in a list (dvList) or in a grid (dvGrid)

TWebGoogleMaps



Description

Below is a list of the most important properties, methods and events for TWebGoogleMaps.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebGoogleMaps

APIkey	Sets the Google Maps JavaScript API key
ElementClassName	Optionally sets the CSS classname for the map when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new map is created but the Delphi class is connected with the existing HTML element in the form HTML file
Options	
MapStyle	Sets the style used to display the map. Options are mstDefault, mstNightMode, mstCustom. If set to mstCustom the style specified in CustomStyle is used. Custom styles can be generated at: https://mapstyle.withgoogle.com/
CustomStyle	Sets the custom style that is used when MapStyle is set to mstCustom
DefaultLatitude	Sets the default latitude position of the map
DefaultLongitude	Sets the default longitude position of the map
DefaultZoomLevel	Sets the default zoom level of the map
Markers[AIndex: Integer] : TJSTObject	Array of markers currently displayed on the map
Polygons[AIndex: Integer] : TJSTObject	Array of polygons currently displayed on the map
Polylines[AIndex: Integer] : TJSTObject	Array of polylines currently displayed on the map

Circles[AIndex: Integer] : TJLObject	Array of circles currently displayed on the map
Rectangles[AIndex: Integer] : TJLObject	Array of rectangles currently displayed on the map

Methods for TWebGoogleMaps

SetCenter(Lat, Lon: Double);	Centers the map around geocoordinate Lat/Lon
SetZoom(Zoom: Integer);	Controls the map zoom level (between 1 and 21 for US & Europe, other areas the maximum zoom level might be lower)
AddMarker(Lat, Lon: Double; Title: string = "");	Adds a marker with optional title at geocoordinate Lat/Lon
AddMarker(Lat, Lon: Double; PinIcon: string; Title: string = "");	Adds a marker with image URL PinIcon and with optional title at geocoordinate Lat/Lon
AddMarker(Lat, Lon: Double; Color: TColor; PinLetter: string; Title: string = "");	Adds a marker with specified color and letter in the pin and with optional title at geocoordinate Lat/Lon
AddMarker(Lat, Lon: Double; Color: TGoogleMarkerColor; Title: string = "");	Adds a default Google marker with specified color and with optional title at geocoordinate Lat/Lon The default Google colors can be: mcDefault, mcRed, mcBlue, mcGreen, mcPurple, mcYellow
AddMarker(Lat, Lon: Double; Shape: TGoogleMarkerShape; Color: TColor; BorderColor: TColor; Scale: Double; CustomShape = string = ""; Title: string = "");	Adds a marker with specified shape, color, bordercolor, scale and with optional title at geocoordinate Lat/Lon The shape can be: msPin, msPinDot, msFlag, msBookmark, msFlagSmall, msHome, msFavorite, msStar, msCustom If msCustom is selected a CustomShape value can be provided.
AddMarker(Lat, Lon: Double; PinIcon: string; Title: string; XOffset: integer = 0; YOffset: integer = 0);	Adds a marker with specified image URL and hint at geocoordinate Lat/Lon. Optionally, an X,Y offset of the image versus the Lat/Lon position can be specified

AddPolyline(Points: TJSArray; AColor: TColor = clRed; AWidth: Integer = 2; AOpacity: Double = 1)	Adds a polyline with the specified coordinate Points and with optional color, width and opacity
AddPolygon(Points: TJSArray; AFillColor: TColor = clRed; AStrokeColor: TColor = clBlack; AWidth: Integer = 2; AOpacity: Double = 1)	Adds a polygon with specified coordinate Points and with optional fill color, stroke color, width and opacity
AddCircle(Lat, Lon: Double; Radius: Integer; AFillColor: TColor = clRed; AStrokeColor: TColor = clBlack; AWidth: Integer = 2; AOpacity: double = 1)	Adds a circle with specified center coordinates, radius and optional fill color, stroke color, width and opacity
AddRectangle(NorthEastLat, NorthEastLon, SouthWestLat, SouthWestLon: Double; AFillColor: TColor = clRed; AStrokeColor: TColor = clBlack; AWidth: Integer = 2; AOpacity: Double = 1)	Adds a rectangle with specified coordinates, radius and optional fill color, stroke color, width and opacity
AddGPX(AGPX: string; AColor; TColor; AWidth: Integer; AOpacity: Double);	Adds a GPX layer with optional Color, Width and Opacity to the map
AddKML(Url: string; ZoomToBounds: Boolean = true)	Adds a KML layer with specified Url to the map and optionally zoom to the KML layer bounds
ClearMarkers	Removes all markers from the map
ClearPolylines	Removes all polylines from the map
ClearPolygons	Removes all polygons from the map
ClearCircles	Removes all circle from the map
ClearRectangles	Removes all rectangles from the map
ClearKMLs	Removes all KMLs from the map
ShowDirections(Source, Destination: string; ATravelMode: TGoogleTravelMode = tmDriving; WayPoints: TStringList = nil; OptimizeWayPoints: Boolean = False; AvoidHighways: Boolean = False; AvoidTolls: Boolean = False);	Show the calculated route between Source and Destination expressed as addresses. Optionally set TravelMode, add WayPoints, OptimizeWayPoints, AvoidHighways, AvoidTolls
ShowDirections(SourceLon, SourceLat, DestLon, DestLat: Double; ATravelMode: TGoogleTravelMode = tmDriving; WayPoints: TStringList = nil; OptimizeWayPoints: Boolean = False; AvoidHighways: Boolean = False; AvoidTolls: Boolean = False);	Show the calculated route between Source and Destination expressed as coordinates. Optionally set TravelMode, add WayPoints, OptimizeWayPoints, AvoidHighways, AvoidTolls
RemoveDirections	Removes the display of a route on the map

GeoCode(const Address: string);	Converts the address to the geocoordinate Lat/Lon. The result of the conversion is retrieved via the event OnGeoCoded
PanTo(Lat, Lon: Double)	Pan the center of the map to the provided coordinates
SetZoom(Zoom: Integer)	Zoom the map to the provided zoom level
FitBounds(LatMin, LonMin, LatMax, LonMax: Double)	Pan and zoom the map to the bounds of the provided coordinates
GetCenter(var Lat, Lon: Double): Boolean	Returns the current center coordinate of the map
GetBounds(var NorthEastLat, NorthEastLon, SouthWestLat, SouthWestLon: Double): Boolean;	Returns the current bounds of the map
SetDoubleClickZoom(AValue: Boolean)	Sets if the map is zoomed when a double click occurs
SetScrollWheel(AValue: Boolean)	Sets if the map is zoomed when the mouse wheel is used
SetDraggable(AValue: Boolean)	Sets if the map can be dragged to a new position
SetMapType(AMapType: TGoogleMapType = mtDefault)	Sets the map type to display. Options are mtDefault, mtSatellite, mtHybrid, mtTerrain
SetMarkerTitle(AIndex: Integer; ATitle: string)	Sets the title of the marker with index AIndex
SetMarkerLocation(AIndex: Integer; Lat, Lon: Double);	Sets the location of the marker with index AIndex
SetMarkerIcon(AIndex: Integer; Url: string);	Sets the icon of the marker with index AIndex
SetCircleCenter(AIndex: Integer; Lat, Lon: Double);	Sets the center of the circle with index AIndex
SetCircleRadius(AIndex, Radius: Integer);	Sets the radius of the circle with index AIndex
SetCircleColors(AIndex: Integer; AFillColor, AStrokeColor: TColor);	Sets the colors of the circle with index AIndex
SetRectangleLocation(AIndex: Integer; NorthEastLat, NorthEastLon, SouthWestLat, SouthWestLon: Double);	Sets the location of the rectangle with index AIndex
SetRectangleColors(AIndex: Integer; AFillColor, AStrokeColor: TColor);	Sets the colors of the rectangle with index AIndex
SetPolylineColor(AIndex: Integer; AColor: TColor);	Sets the color of the polyline with index AIndex
SetPolylinePoints(AIndex: Integer; Points:	Sets the points of the polyline with index

TJSArray);	AIndex
SetPolygonColors(AIndex: Integer; AFillColor, AStrokeColor: TColor);	Sets the colors of the polygon with index AIndex
SetPolygonPoints(AIndex: Integer; Points: TJSArray);	Sets the points of the polygon with index AIndex
ShowStreetView(Lat, Lon: Double; Heading: Integer = 0; Zoom: Integer = 0; Pitch: Integer = 0)	Display streetview mode for the provided coordinates. Optionally set the heading direction, zoom level and pitch value
HideStreetView	Hide streetview mode
RemoveMarker(AIndex: Integer);	Remove the marker with index AIndex from the map
RemovePolygon(AIndex: Integer)	Remove the polygon with index AIndex from the map
RemovePolyline(AIndex: Integer);	Remove the polyline with index AIndex from the map
RemoveCircle(AIndex: Integer);	Remove the circle with index AIndex from the map
RemoveRectangle(AIndex: Integer);	Remove the rectangle with index AIndex from the map

Events for TWebGoogleMaps

OnCircleClick	Event triggered when a Circle is clicked
OnGeoCoded	Event triggered when the geocoding started with WebGoogleMaps.GeoCode() was successful
OnKMLClick	Event triggered when a KML is clicked
OnMarkerClick	Event triggered when a marker is clicked
OnMapClick	Event triggered when the map is clicked
OnMapDbClick	Event triggered when the map is double-clicked
OnMapIdle	Event triggered when map interaction has ended
OnMapLoaded	Event triggered when the map has finished loading. Note that Properties/Methods that interact with the map should only be used after this event was triggered.

OnMapPan	Event triggered when the map is panned
OnMapZoom	Event triggered when the map is zoomed
OnPolylineClick	Event triggered when a Polyline is clicked
OnPolygonClick	Event triggered when a Polygon is clicked
OnRectangleClick	Event triggered when a Rectangle is clicked

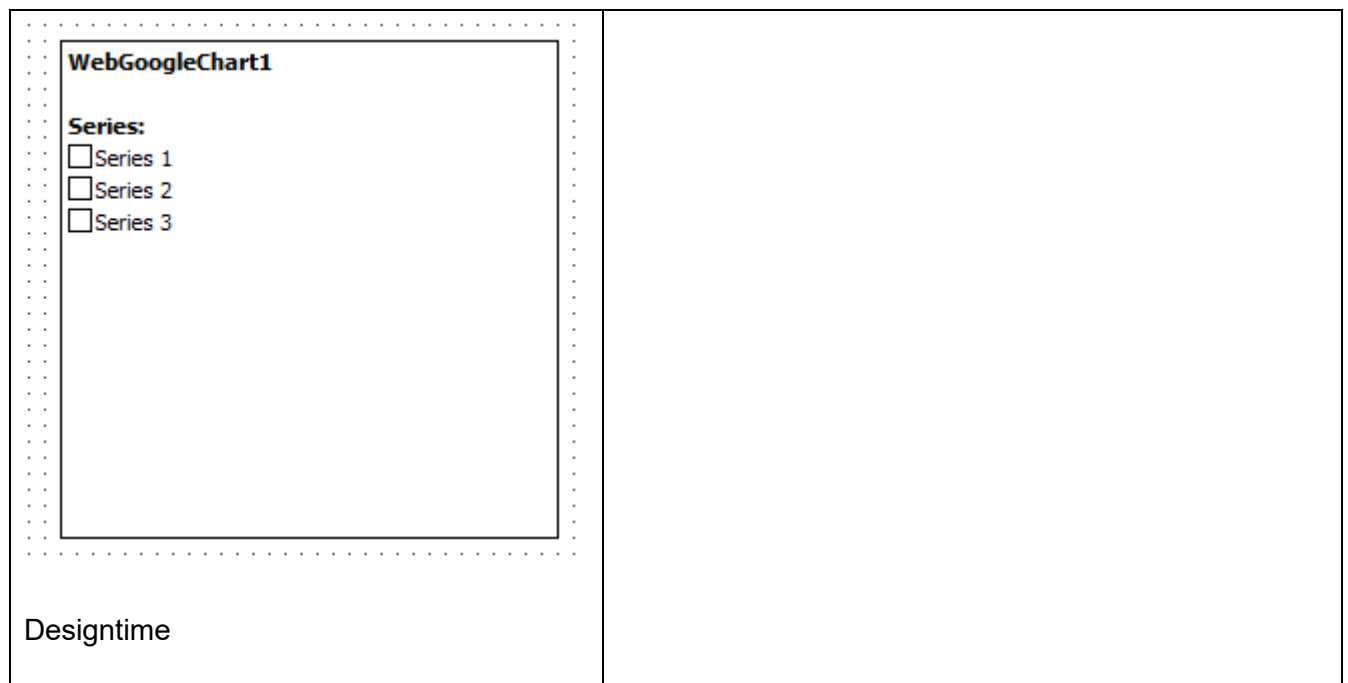
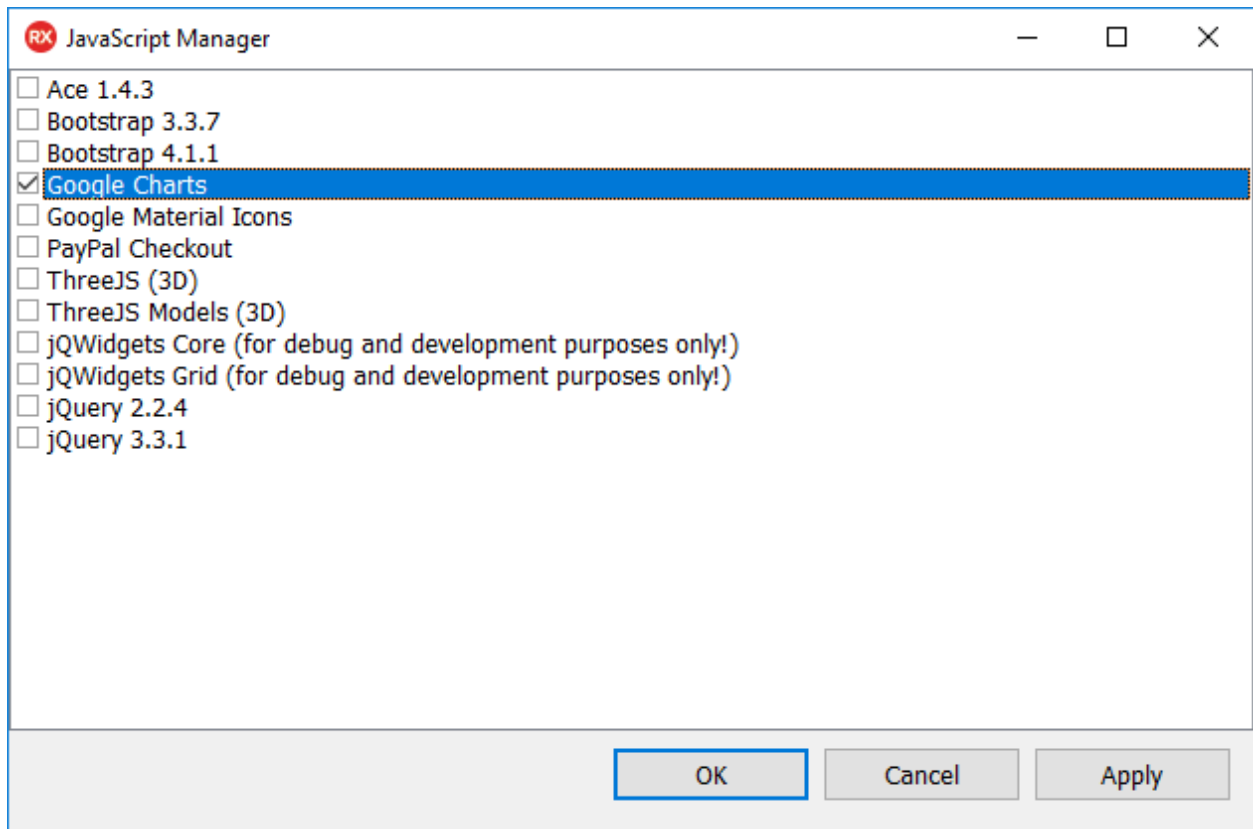
TWebGoogleChart

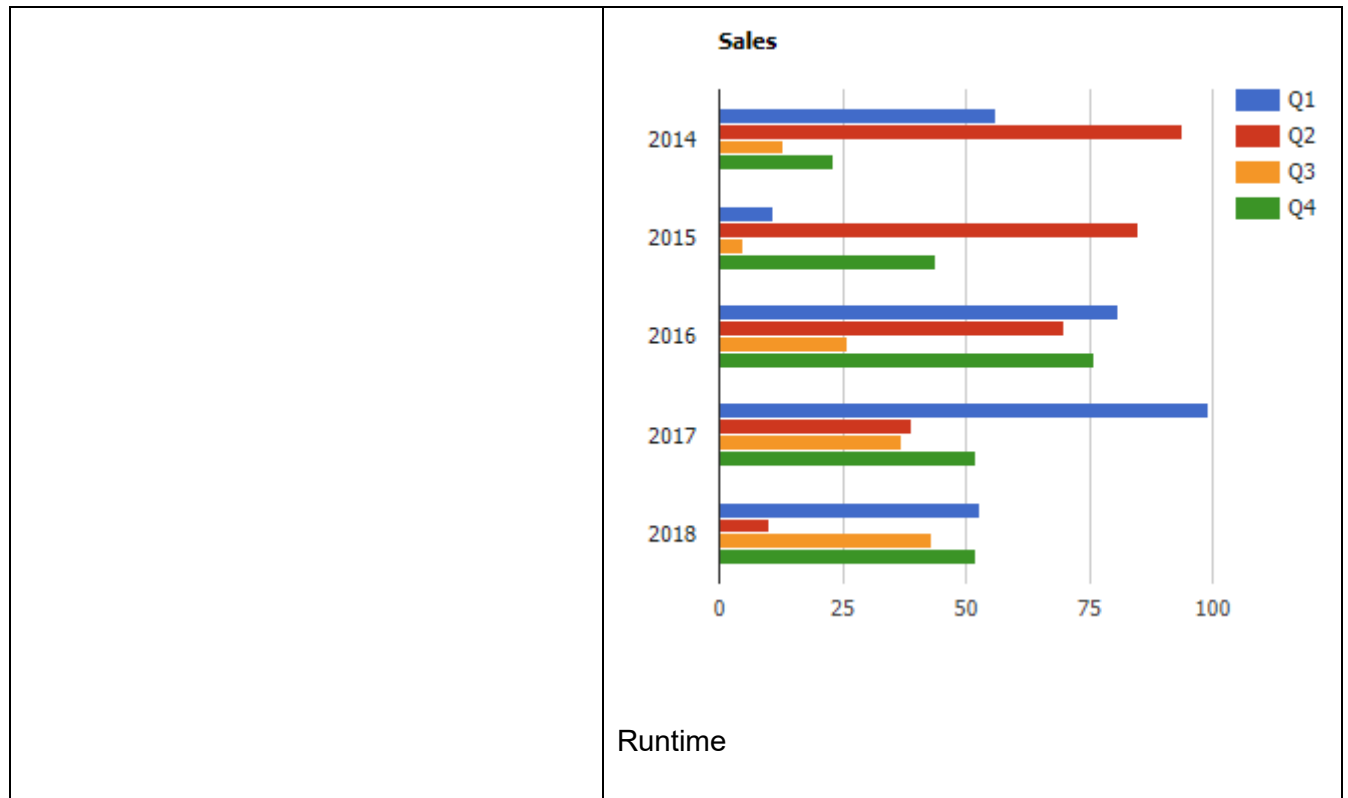


Description

Below is a list of the most important properties methods and events for TWebGoogleChart.

Note: To use Google Charts, it is important to activate the needed Google Charts JavaScript library for this. Do this from the “Manage JavaScript Libraries” item from the project context menu in the IDE project manager.





Properties for TWebGoogleChart

Appearance

Animation.Duration	Duration of the chart animation
Animation.Easing	Sets the type of animation easing.
Animation.Startup	Sets if the chart is animated on startup
Background.BorderColor	Sets the border color
Background.BorderWidth	Sets the border width
Background.Color	Sets the background color
HAxis.AutoMaxMinValue	Automatically set the Max and Min values of the HAxis based on the point values. If true the MaxValue and MinValue properties are ignored
HAxis.MaxValue	Sets a custom Max value for the HAxis
HAxis.MinValue	Sets a custom Min value for the HAxis
Legend.Alignment	Sets the alignment of the legend
Legend.Position	Sets the position of the legend
LineChart.CurveType	Sets the curve of the line. Set to None to

	disable line curve or Function to enable
PieChart.Enable3D	Sets if a chart of type Pie is displayed in 3D
PieChart.PieHole	Displays the Pie chart as a Donut chart. The value configures the size of the donut hole. Ignored if Enable3D is true
PieChart.PieSliceText	Sets which data is displayed on each pie slice. Options are Label, None, Percentage or Value
ReverseCategories	Sets the order in which the categories are added to the chart. 1 for default order, 0 for reversed order
Stacked	Sets if data in a Bar, Column or Area chart is displayed stacked or not
Tooltip	Configures when the tooltip is displayed
VAxis.AutoMaxMinValue	Automatically set the Max and Min values of the VAxis based on the point values. If true the MaxValue and MinValue properties are ignored
VAxis.MaxValue	Sets a custom Max value for the VAxis
VAxis.MinValue	Sets a custom Min value for the VAxis
Chart	Returns the chart as a TJXObject. Allows customizing the chart via JavaScript calls after the initial rendering. (See Example 3)
Data	Returns the chart data as TJXObject. Allows customizing the chart data via JavaScript calls after the initial rendering. (See Example 3)
Series	
ChartType	Sets the type of chart to display. Only series with ChartType Bar, Column, Area or Line can be combined on a single Chart.
Color	Sets the color of the datapoints
Line.LineWidth	Sets line width for Series of ChartType Area, Line or Scatter. Set to 0 to hide the line and only display points.
Line.PointShape	Sets the shape of the datapoints for Series of ChartType Area, Line or Scatter.

Line.PointSize	Sets the size of the datapoints for Series of ChartType Area, Line or Scatter. Set to 0 to hide the points and only display lines.
Title	Sets the title of the Series
Title	Sets the title of the chart

Methods for TWebGoogleChart

SetOption(AOption: string; AValue: Boolean); SetOption(AOption: string; AValue: TJSObject); SetOption(AOption: string; AValue: string);	Changes a chart option after the chart is rendered. (See Example 4)
Series[].Values.AddSinglePoint(AValue: Double; ALabel: string = "");	Adds a point to a chart of type Area, Bar, Column, Line.
Series[].Values.AddPiePoint(AValue: Double; ALabel: string = ""; Offset: Double = 0; Color: TColor = clNone);	Adds a point to a chart of type Pie. The Offset parameter sets the distance of the pie slice from the main pie. The Color sets the backgroundcolor of the slice, set to clNone to use default colors.
Series[].Values.AddXYPoint(X, Y: Double);	Adds a point to a chart of type Scatter.
Series[].Values.AddCandlestickPoint(X, Y, Minimum, Maximum: Double; ALabel: string = "");	Adds a point to a chart of type Candlestick.
Series[].Values.AddTimelinePoint(StartTime, EndTime: TDateTime; ALabel: string = "");	Adds a point to a chart of type Timeline.
Series[].Values.AddBubblePoint(X, Y: Double; Series: string; Size: Double; ALabel: string = "");	Adds a point to a chart of type Bubble.
Series[].Values.AddBubbleColorPoint(X, Y: Double; Value: Double; ALabel: string = "");	Adds a point to a chart of type BubbleColor. The Value parameter determines the color of the bubble.

Events for TWebGoogleChart

OnLoaded(Sender: TObject);	Event triggered when the has finished loading
OnSelect(Sender: TObject; Event: TGoogleChartSelectEventArgs);	Event triggered when a datapoint on the chart is selected. The Event parameter contains the SeriesIndex and the PointIndex
OnCustomizeChart(Sender: TObject; var Options: TGoogleChartOptions);	Event triggered before the chart rendering starts. Allows configuration of selected extended chart properties via the Options parameter values. (See Example 2)
OnCustomizeChartJSON(Sender: TObject; var Options: string);	Event triggered when the chart configuration JSON data is ready. Allows to fully customize the configuration of the chart via the Options parameter.

Examples

Example 1: Configuring a BarChart

Demonstrates how to display a chart with just a few lines of code.

```
var
  it: TGoogleChartSeriesItem;
begin
  it := WebGoogleChart1.Series.Add;
  it.ChartType := gctPie;
  it.Values.AddPiePoint(80, 'Label A');
  it.Values.AddPiePoint(20, 'Label B');
end;
```

Example 2: Customization options

Demonstrates how to customize a chart with extended options.

```
procedure TForm1.WebGoogleChart1CustomizeChart(Sender: TObject;
  var Options: TGoogleChartOptions);
begin
  Options.HAxis.ViewWindow.Min := '0';
  Options.HAxis.ViewWindow.Max := '100';
end;
```

Note: Options data must contain valid JSON data.

Full documentation of available configuration options can be found at:

<https://developers.google.com/chart/interactive/docs/>

(Select Chart Type from the list on the left, then select “Configuration Options” from the “Content” items on the right)

Example 3: Adding and updating datapoints on the fly

Demonstrates how to dynamically add and update datapoints in an existing chart.

```
procedure TForm1.WebButton1Click(Sender: TObject);
var
    data: TJSObject;
    chart: TJSObject;
begin
    data := WebGoogleChart1.Data;
    chart := WebGoogleChart1.Chart;
    asm
        data.setValue(0, 1, 20); //rowIndex, columnIndex, value
        data.addColumn('number', 'Label'); //datatype, label
        data.addRow(['Row', 10, 20, 30, 40]); //rowTitle, Column values
        chart.draw(); //update chart
    end;
end;
```

Note: The Google Charts API reference can be found here:

<https://developers.google.com/chart/interactive/docs/reference#methods>

Example 4: Setting options on the fly

Demonstrates how to dynamically update options in an existing chart.

```
procedure TForm1.WebButton1Click(Sender: TObject);
begin
    WebGoogleChart1.SetOption('vAxis.title', 'Y axis title');
end;
```

Note: Options data must contain valid JSON data.

Full documentation of available configuration options can be found at:

<https://developers.google.com/chart/interactive/docs/>

(Select Chart Type from the list on the left, then select “Configuration Options” from the “Content” items on the right)

TWebSentry



Sentry.io is a cloud-based error monitoring service that can log errors from your Web App even when it is being used by the customers.

Each error is logged as an issue and you can see the Stack Trace for each issue that can help diagnose the problem.

Once the issues are logged, the Sentry dashboard has convenient features to manage these issues, for example, to assign them to other users who can see their issues and so on.

TMS Web Core provides a component “TWebSentry” that integrates Sentry.io with your web core application. It encapsulates all the logic of sending errors to Sentry so that they are logged as Issues. Also, the issues logged by TWebSentry in Sentry contain a Stack Trace that conveniently shows the Delphi Pascal code.

Steps to set up

Sign up with Sentry.io

You can get started for Free.

Please go to <https://sentry.io/auth/login/> and sign in with Google. It will ask you to sign up as a New Organization.

Select an organization name and proceed to set up the account.

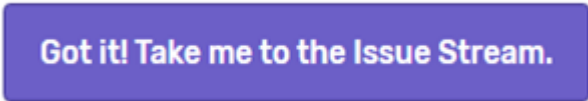
Perform these steps in the Dashboard

Create a project.

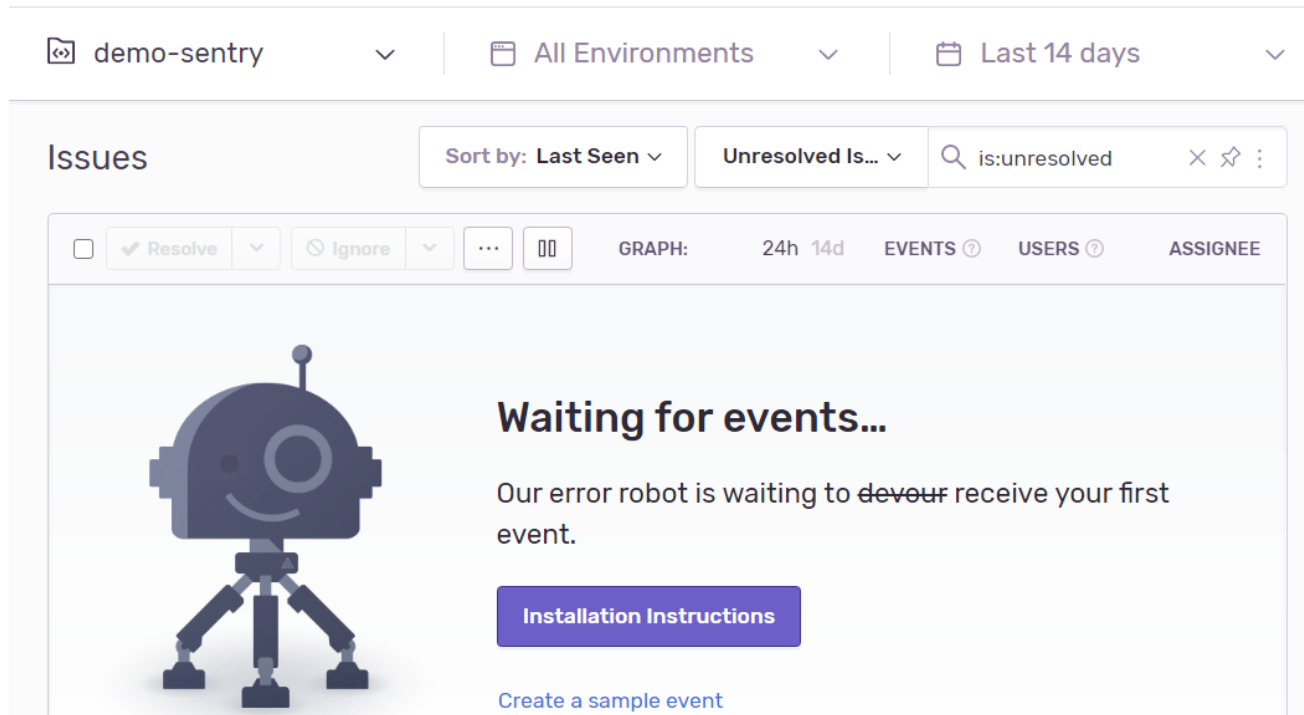
Select JavaScript as the platform.

Enter a project name.

On adding the project, it will display a screen of instructions. Please ignore them as the Sentry web core component will be doing all that for you. Scroll to the bottom and you will see a button “Take me to the Issue Stream.”



Click the button and it will show the Issues screen saying “Waiting for verification event.”



Note that later you will be reaching the same Issues screen often from the “Issues” menu on the left.

The event will complete when you follow the steps given below to set up your Delphi Web Core App so that its errors end up as Issues on this screen.

Open DemoSentry project in Delphi

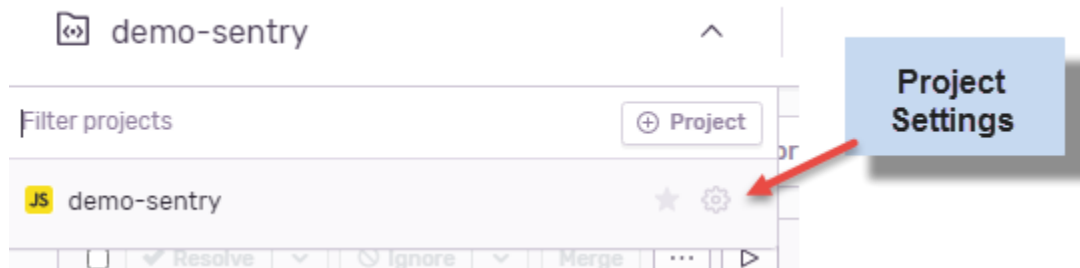
For the purpose of Demonstrating TWebSentry, there is a DemoSentry project in TMS Web Core. In the following discussion, we give steps to use this Demo to see various features.

Copy required parameters from the Dashboard to Paste in Sentry Demo

The dashboard screens below use an organization name as “tms-software” and the project name as “demo-sentry”. But you can select any other names and it will still work with DemoSentry as long as the following steps are completed properly.

First bring up Project Settings

To do that, click on the drop down next to project name at the top.



Click on the Gear icon next to the project as shown above.
Select “Client Keys (DSN)” on the Settings menu under SDK SETUP.
Copy the value of the DSN box by using the button next to it.

Paste the DSN in the DemoSentry project as given below

Open the source code of the unit USentry.pas.
Paste the DSN value for the DSN property of the component in WebFormCreate.
Now Build the project and Run it. You will see the following screen in the browser.

The screenshot shows a web browser window at localhost:8000/TMSWeb_Sentry/index.html. The page has a header with the 'tms' logo, a 'WEB' tab, and 'TMS WEB Core' text. A decorative graphic of a person running is on the right. Below the header is a teal banner with the title 'Sentry error logging demo'. An information icon and text state: 'Logs JavaScript errors and Object Pascal exceptions to Sentry service'. A checkbox 'Send to Sentry Enabled' is checked. The first section, 'Code sample: Send anticipated (caught) errors to Sentry log', contains two buttons: 'Catch Delphi Error and Send to Sentry' and 'Catch JS Error and Send to Sentry'. To the right is a text input field with the placeholder 'Remark (optional) appears as Additional Data'. The second section, 'Simulate unexpected (uncaught) errors to demonstrate "Auto Send" to Sentry log', contains two buttons: 'Raise Delphi Exception Uncaught' and 'Throw JS Error Uncaught'. The third section, 'Code sample: Send a message to Sentry log', contains a button 'Send Log Message to Sentry' and a text input field with the placeholder 'Message' and the value 'Hello from TMS Web Core'.

The purpose of this Demo is to create a variety of error types to see how they appear in Sentry Dashboard as issues. You can always look at the Form code to see the actual sample code.

Let's raise a Delphi Exception, catch it and send to Sentry

We will force a Delphi error in code, catch it in an Exception block and then send it to Sentry by a `CaptureException` call.

To do that, click on the button "Catch Delphi Error and Send to Sentry" that executes the following code.

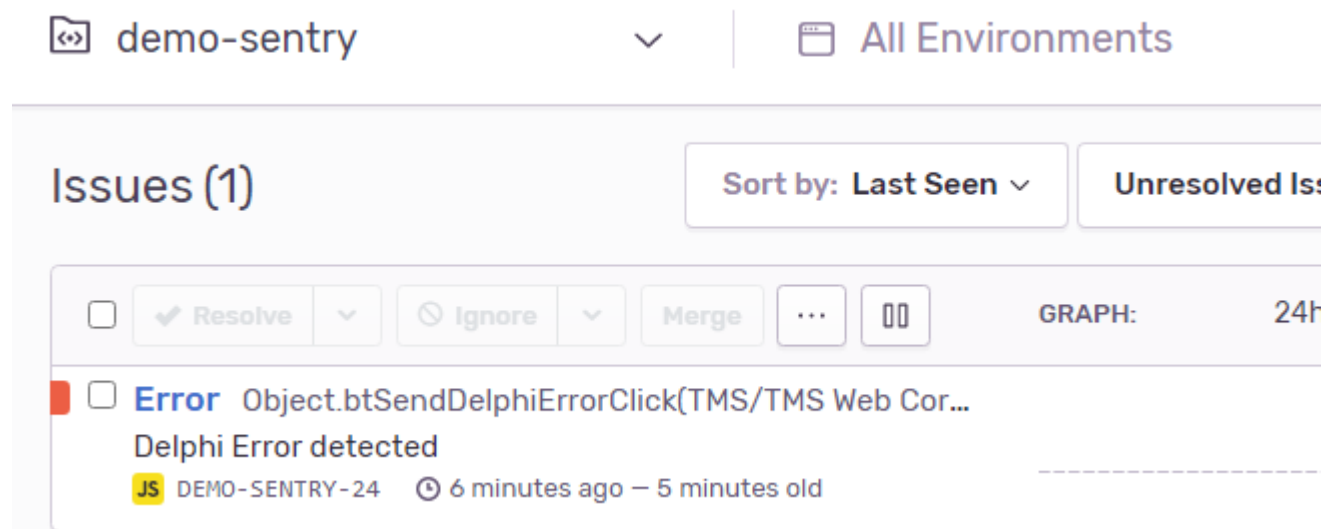
```
procedure TForm1.btSendDelphiErrorClick(Sender: TObject);
var
    sendException: TJSObject;
begin
    try
        raise Exception.Create('Delphi Error detected');
    except
        On E: Exception do
        begin
            sendException := TJSObject(E);
        end;
    end;
    WebSentry1.CaptureException(sendException, edRemark.Text);
end;
```

As you can see, the exception is sent by calling method `CaptureException` of `WebSentry1` component. The second parameter is an optional remark that we can fill up and send.

Now switch to the Sentry dashboard in the browser

We want to see if an Issue is recorded for the Delphi exception.

If you see the Issues screen that was waiting for an event, you should see an Issue now.



The second line above shows the error message that we raised the Delphi Exception with in earlier code.

This is great because you will be able to see the errors from your TMS Web Core App in Sentry as issues

Further, you will be able to see the errors no matter where the customer is using the App.

What is even more useful is that you will be able to see the Stack Trace at the time of error

Click on the above Issue to see the details. Scroll down a little and you should see the Stack Trace.

Error

Delphi Error detected

mechanism generic handled yes

```
JS /DemoSentry/DemoSentry.js in Object.btSendDelphiErrorClick at line 37373:38
37368.     this.WebSentry1.Init();
37369. };
37370.     this.btSendDelphiErrorClick = function (Sender) {
37371.         var sendException = null;
37372.         try {
37373.             throw pas.SysUtils.Exception.$create("Create$1",["Delphi Error detected"]);
37374.         } catch ($e) {
37375.             if (pas.SysUtils.Exception.isPrototypeOf($e)) {
37376.                 var E = $e;
37377.                 sendException = E;
37378.             } else throw $e

/DemoSentry/DemoSentry.js in Object.cb [as FOnClick] at line 226:19
/DemoSentry/DemoSentry.js in Object.Click at line 24204:61
/DemoSentry/DemoSentry.js in Object.HandleDoClick at line 23812:12
/DemoSentry/DemoSentry.js in HTMLButtonElement.cb at line 222:26
```

But you will notice that this Stack Trace shows the JS code. That's not so useful. Why don't we see the Pascal code?

The reason is that the demo is running on localhost which Sentry can not access. If you copy the files of this demo to a web site and then follow the same steps as above to produce an issue, you will see the Delphi code with Pascal Stack Trace.

Let's get the proper Pascal Stack Trace by running this demo from a web location

Go to the httdocs directory on your computer where the Web App is created by the build. From there, copy the output files to a web host. Now run the same Demo from there and produce the same issue. If you see the stack trace for that issue, you should see the proper Pascal stack trace as in the following screenshot.

Error

Delphi Error detected

mechanism

generic

handled

yes

JS

[../../TMS/TMS Web Core/Dev/Sentry/Demo/USentry.pas](#) in [Object.btSendDelphiErrorClick](#) at line 66:20

```

61. procedure TForm1.btSendDelphiErrorClick(Sender: TObject);
62. var
63.   sendException: TJSError;
64. begin
65.   try
66.     raise Exception.Create('Delphi Error detected');
67.   except
68.     On E: Exception do
69.     begin
70.       sendException := TJSError(E);
71.     end;

```

[../../TMS/TMS Web Core/Core Source/RTL/rtl.js](#) in [Object.cb \[as FOnClick\]](#) at line 221:1

[../../TMS/TMS Web Core/Core Source/WEBCoreLib.Controls.pas](#) in [Object.Click](#) at line 1508:5

[../../TMS/TMS Web Core/Core Source/WEBCoreLib.Controls.pas](#) in [Object.HandleDoClick](#) at line 3196:3

[../../TMS/TMS Web Core/Core Source/RTL/rtl.js](#) in [HTMLButtonElement.cb](#) at line 217:1

See how the correct source line that raises Delphi exception is shown from the Pascal unit USentry.pas.

More on the Source Map file

If you see the httdocs folder for DemoSentry project where a build operation creates the output files for the Web App, you will 2 JS files.

DemoSentry.js

DemoSentry.js.map

The map file is the source map file that is needed to show the proper stack trace in Pascal. When you uploaded the Web App to a web host above and ran the Web App from there, Sentry could access the map file and could log the proper Pascal stack trace with the issue. Note that currently the map file is only created when the project is built in Debug configuration.

An option to create the map file should be there for Release configuration too so that Web Apps in production also get this feature to send Pascal stack trace.

Security Problem with the hosted Map file

Putting the source map file on the web host is a security risk because then it can be accessed publicly and seen with all the code for the App!

The solution is to upload the source map file to Sentry and remove it from the web host so that it is not publicly available and is only available to Sentry when logging its issues.

This is an advanced operation and requires you to install a command line tool called Sentry-cli on your system. Please refer to the Sentry documentation to see how to download and install Sentry-cli.

Using Sentry-cli to upload MAP file to Sentry

Let's assume that you hosted the DemoSentry files from C: to the following web location:

<https://mytest.com/DemoSentry>

If so, the command line to upload the map file to Sentry is:

```
sentry-cli releases --org tms-software --project demo-sentry files  
"DemoSentry@1.0" upload-sourcemaps C:\htdocs\DemoSentry --url-prefix  
https://mytest.com/DemoSentry --rewrite
```

Where tms-software is the organization from the dashboard, demo-sentry is the project name from the dashboard and "DemoSentry@1.0" is the Release from the Delphi source file. Release is explained in the next section.

The above command uploads both the JS and JS Map file to Sentry. Then you can remove only the Map file from the hosted web app and the stack trace will still appear properly with the issues logged after that.

What is a release

You will notice a property Release set up for the WebSentry1 component in the Delphi USentry unit source along with the DSN.

In Sentry, the issues are always under a release. This is quite logical because once you do bug fixes for errors, you are creating another release of your app. In that case, you should change the Release value in the Delphi source file. That way the errors (issues) related to a different releases are kept separate.

For the same reason, the source maps are also associated with a Release. So when you have another Release, you will need to upload your new Source Map files under the new release.

Continuing with the rest of the Demo

1. Catch Delphi Error and Send to Sentry


We have already seen this case of sending a “caught” Delphi error to Sentry and inspecting the logged issue along with the stack trace in Sentry dashboard.

There are 3 other error conditions demonstrated in the Sentry Demo.

2. Catch JS Error and Send to Sentry

To see this in action, click on the button “Catch HS Error and Send to Sentry.”

Sentry error logging demo

 Logs JavaScript errors and Object Pascal exceptions to Sentry service

☒ Send to Sentry Enabled

Code sample: Send anticipated (caught) errors to Sentry log

Catch Delphi Error and Send to Sentry

Catch JS Error and Send to Sentry

Remark (optional) appears as Additional Data

Simulate unexpected (uncaught) errors to demonstrate "Auto Send" to Sentry log

Raise Delphi Exception Uncaught

Throw JS Error Uncaught

Code sample: Send a message to Sentry log

Send Log Message to Sentry

Message

You can try this, look at the code sample and see how the issue and stack trace appears in Sentry.

Error

JS Error detected

mechanism

generic

handled

yes

JS

../TMS/TMS Web Core/Dev/Sentry/Demo/USentry.pas in Object.btSendJSErrorClick at line 82:1

```

77. var
78.   sendException: TJSObject;
79. begin
80.   asm
81.   try {
82.     throw new Error('JS Error detected');
83.   }
84.   catch(err) {
85.     sendException = err;
86.   }
87. end;

```

../TMS/TMS Web Core/Core Source/RTL/rtl.js in Object.cb [as FOnClick] at line 221:1

../TMS/TMS Web Core/Core Source/WEBLib.Controls.pas in Object.Click at line 1508:5

../TMS/TMS Web Core/Core Source/WEBLib.Controls.pas in Object.HandleDoClick at line 3196:3

../TMS/TMS Web Core/Core Source/RTL/rtl.js in HTMLButtonElement.cb at line 217:1

As you can see, the call stack is correct, pointing to the proper line in the source that throws the error.

How to log an additional remark along with the Exception

If you see the code that calls `CaptureException` in the unit, you will see a second parameter that can send an optional Remark string to be logged in the issue.

```

procedure TForm1.btSendDelphiErrorClick(Sender: TObject);
var
    sendException: TJSObject;
begin
    try
        raise Exception.Create('Delphi Error detected');
    except
        On E: Exception do
            begin
                sendException := TJSObject(E);
            end;
        end;
    WebSentry1.CaptureException(sendException, edRemark.Text);
end;

```

To see this in action, enter some text in the Remark text box before you click on the Catch Delphi Error button. Then if you see the details of the newly logged issue in the Sentry Dashboard, you will see the Remark in the Additional Data section further down the page as shown in the following screenshot.

ADDITIONAL DATA

FromDelphi	yes
Remark	Testing remark feature
UncaughtException	False

What happens if Unexpected Errors occur in the Web App?

The cases that we saw earlier are anticipated errors that we catch and send to Sentry by calling `CaptureException`.

What happens when unexpected errors occur either in your Web App or in Web Core? They are automatically sent to Sentry to be logged as issues.

This feature is demonstrated with the second group of buttons in the Demo under “Simulate unexpected (Uncaught) errors.”

Sentry error logging demo

 Logs JavaScript errors and Object Pascal exceptions to Sentry service

☒ Send to Sentry Enabled

Code sample: Send anticipated (caught) errors to Sentry log

Catch Delphi Error and Send to Sentry

Catch JS Error and Send to Sentry

Remark (optional) appears as Additional Data

Simulate unexpected (uncaught) errors to demonstrate "Auto Send" to Sentry log

Raise Delphi Exception Uncaught

Throw JS Error Uncaught

Code sample: Send a message to Sentry log

Send Log Message to Sentry

Message

Hello from TMS Web Core

3. Raise Delphi Exception Uncaught

Just click on the button "Raise Delphi Exception Uncaught." The code just raises a Delphi exception to simulate this condition. It doesn't catch it or call any Sentry method.

Still, the error is reported to Sentry properly.

See the corresponding issue and the call stack in Sentry dashboard.

Unexpected Delphi Error (uncaught)

mechanism	generic	handled	yes
-----------	---------	---------	-----

JS

../TMS/TMS Web Core/Dev/Sentry/Demo/USentry.pas in Object.btRaiseJSErrorClick at line 104:19

```
99.     WebSentry1.CaptureException(sendException, aRemark);
100. end;
101.
102. procedure TForm1.btRaiseJSErrorClick(Sender: TObject);
103. begin
104.     raise Exception.Create('Unexpected Delphi Error (uncaught)')
105. end;
106.
107. procedure TForm1.btRaiseDelphiErrorClick(Sender: TObject);
108. begin
109.     asm
```

../TMS/TMS Web Core/Core Source/RTL/rtl.js in Object.cb [as FOnClick] at line 221:1

../TMS/TMS Web Core/Core Source/WEBCLib.Controls.pas in Object.Click at line 1508:5

Isn't this wonderful? This means you don't even need to modify your Web App. Just use the `WebSentry` component as described above and you get this feature out-of-the-box. Any Delphi exceptions occurring in your code or in Web Core on the Customer locations will be reported as issues in the Sentry dashboard.

Similarly if you click on the second button, it throws a JS Error that is automatically sent to Sentry and logged as an issue with the following stack trace.

Unexpected JS Error (uncaught)

no

—



Sometimes, even without errors, you may want to log an informational message in Sentry log.

Sentry error logging demo

 Logs JavaScript errors and Object Pascal exceptions to Sentry service

☒ Send to Sentry Enabled

Code sample: Send anticipated (caught) errors to Sentry log

Catch Delphi Error and Send to Sentry

Catch JS Error and Send to Sentry

Remark (optional) appears as Additional Data

Simulate unexpected (uncaught) errors to demonstrate "Auto Send" to Sentry log

Raise Delphi Exception Uncaught

Throw JS Error Uncaught





Code sample: Send a message to Sentry log




Send Log Message to Sentry







Message



Hello from TMS Web Core



This can be done by calling CaptureMessage function of the WebSentry component. In the Demo, click on the button "Send Log Message to Sentry" to do that. It will appear in Sentry issues like the following screenshot.

 demo-sentry  |  All Environments 

Issues (3) Sort by: Last Seen  Unresolved Issues  

☐  Resolve   Ignore  Merge   GRAPH: 24h

☐ **Hello from TMS Web Core**
Object.btSendMessageClick(TMS/TMS Web Core/Dev/Sentry/Demo/USentry)
 DEMO-SENTRY-27  a few seconds ago – a few seconds old

☐ **Error** Object.btSendJSErrorClick(TMS/TMS Web Core/Dev/Sentry/Demo/U...
JS Error detected
 DEMO-SENTRY-26  an hour ago – an hour old

☐ **Error** Object.btSendDelphiErrorClick(DemoSentry/DemoSentry)

Even the stack trace will be there if you look into the issue details.

More features in Sentry

These are not used in the Demo but methods exist in TWebSentry component to use these features.

Set User

Sentry logs issues from all the customers using your Web App. In that case, how can you distinguish an issue coming from a particular user? By default, Sentry logs the ip address as user at the top of the issue.

But you can do better and set a user yourself by calling SetUser method of TWebSentry as soon as you can identify the user, for example, after Login. For example,

```
WebSentry1.setUser('john@example.com');
```

It can be even a user name or id and is entirely upto your app on how you identify the user.

Once that is done, all the issues logged will be under this user.

In Sentry dashboard when you click on an issue, at the top right, you will see how many users are facing this issue. Click on it and you will see the list as shown below.

The screenshot shows a Sentry issue page for a Delphi error. At the top, the error message is "Error Object.btSendDelphiErrorClick(TMS/TMS Web Core/Dev/S...". To the right, it shows "ISSUE # DEMO-SENTRY-29", "EVENTS 3", and "USERS 2". Below the error message, there are buttons for "Resolve", "Ignore", "Star", "Share", and "Feedback". The "Affected Users" section is expanded, showing a table with two users: "johndoe@example.com" (66.67%) and "johndoe@example2.com" (33.33%), both seen 2 minutes ago.

%	User	Last Seen
66.67%	johndoe@example.com	2 minutes ago
33.33%	johndoe@example2.com	2 minutes ago




Breadcrumbs

Sentry supports a concept called Breadcrumbs, which is a trail of events which happened prior to an issue.

For any of the exceptions described above, a breadcrumb appears. When you log a message to console, it also appears as breadcrumb.

In addition, you can call TWebSentry's method AddBreadcrumb to add a breadcrumb with a category and a message. Here is a sample screenshot of breadcrumbs.

BREADCRUMBS

	auth	user logged in	02:53:55
	data	product list obtained successfully	02:53:55
	exception	Error: JS Error detected	02:54:05

The first breadcrumb was added by calling `AddBreadcrumb` with category “auth” and a message to show that a user logged in. Even the user’s identity could have been logged here. Similarly, the second breadcrumb has a category “data” and a message. The third breadcrumb is automatic from the exception that occurred. So the breadcrumbs give us a quick summary that user logged in, data was obtained and then the exception occurred.

Tags

Sentry automatically sets many tags for more details on an issue, for example, browser, os, release, etc. Moreover, the issues can be searched by tags quickly. For example, you can quickly search for issues occurring on OS Windows 7.

You can set custom tags too by calling `SetTag` method of `TWebSentry`. Once you set a tag, it appears on all issues logged after that. Here is an example,

```
WebSentry1.SetTag('ReleaseNote', 'Grid problem fixed.');
```

Here is how the tag appears in the dashboard.

TAGS

ReleaseNote	Grid problem fixed.	browser	Chrome 83.0.4103
browser.name	Chrome	handled	yes
		level	error
		mechanism	generic
os	Windows 7	os.name	Windows
		release	1.0 ⓘ
url	http://localhost:8000/DemoSentry/DemoSentry.html ⓘ		
user	ip:116.75.137.14		

In this case, the tag appears first in the tag cloud.

Properties for TWebSentry

Enabled	When True sends information to Sentry. Set it to False to disable sending it.
DSN	Required. Obtained from Project Settings in Sentry dashboard
Release	Set it to a String that identifies the Release and groups issues under that release. Can be any String, recommended name@version format.

Methods for TWebSentry

Init	<p>If you set DSN and Release property values at design time in Object Inspector then you don't need to call Init explicitly. It's automatically called on loading the form.</p> <p>But if you set DSN and Release property values in code, you must call Init after setting them.</p>
------	--

CaptureException	<pre>procedure CaptureException(anObj: TJSObject; remark: string='');</pre> <p>The component automatically sends Exceptions and Errors to Sentry as long as they are Uncaught.</p> <p>But if you are catching certain Delphi Exceptions or JS Errors in your code, they won't be sent to Sentry for logging unless you explicitly send them by calling CaptureException. Just pass the Delphi Exception object or a JS Error object that you caught as the first parameter to above function. Note that these objects already contain an error message. But if you want to send some additional information, you can send it as a string in the second parameter to CaptureException.</p>
CaptureMessage	<pre>procedure CaptureMessage(aMsg: string);</pre> <p>To send and log an informational message in Sentry log, call CaptureMessage with a string.</p>
SetUser	<p>To better identify issues, set a user soon after you can identify if in the web app, for example, after login. Pass anything for aName that you can identify in the log. For example, it can even be an email address or an id.</p> <pre>procedure SetUser(aName: string);</pre>
AddBreadCrumb	<p>Adds a breadcrumb to be listed in Sentry log.</p> <pre>procedure AddBreadcrumb(aCategory: string; aMessage: string);</pre> <p>Choice of category string is arbitrary. Use anything that makes sense in the logs.</p>


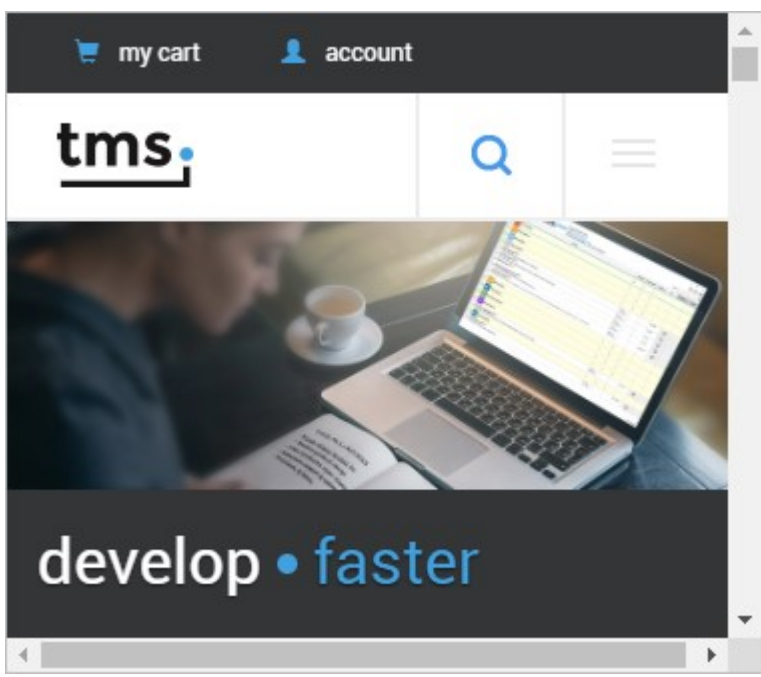
SetTag	<p>Sets a custom tag with a key, value pair that is listed in the issue Tags. The issue also becomes searchable by the tag.</p> <pre>procedure setTag(aKey: string; aValue: string);</pre>
--------	--

TWebBrowserControl



Description

Below is a list of the most important properties methods and events for TWebBrowserControl.

 <p>WebBrowserControl1</p>	
Designtime	Runtime

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<IFRAME ID="UniqueID"></IFRAME>
ElementID	UniqueID

Properties for TWebBrowserControl


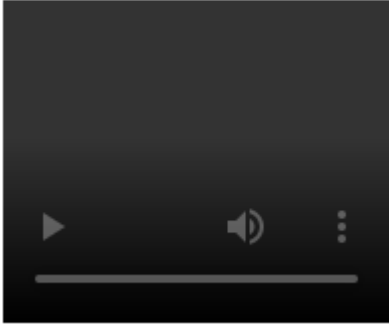
ElementClassName	Optionally sets the CSS classname for the map when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the component needs to be connected with. When connected, no new object is created but the Delphi class is connected with the existing HTML element in the form HTML file
ReferrerPolicy	Sets the preferred referrer policy. Available options are: rfNone, rfNoReferrer, rfNoReferrerWhenDowngrade, rfOrigin, rfOriginWhenCrossOrigin, rfUnsafeUrl
Sandbox	Sets which browse features are allowed. Available options are: stAllowForms, stAllowModals, stAllowOrientationLock, stAllowPointerLock, stAllowPopups, stAllowPopupsToEscapeSandbox, stAllowPresentation, stAllowSameOrigin, stAllowScripts, stAllowTopNavigation, stAllowTopNavigationByUserActivation
URL	Sets the URL to display

TWebMultimediaPlayer



Description

Below is a list of the most important properties methods and events for TWebMultimediaPlayer.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<VIDEO ID="UniqueID"></VIDEO>
ElementID	UniqueID

Properties for TWebMultimediaPlayer

AutoPlay	Sets if the content will starts playing as soon as it is ready
Controls	Sets if the playback controls are displayed
ElementClassName	Optionally sets the CSS classname for the map when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the component needs to be connected with. When connected, no new object is created but the Delphi class is connected with the existing HTML element in the form HTML file
Loop	Sets if the content is played in a continuous

	loop
MultimediaType	Sets if the content is Audio (mtAudio) or Video (mtVideo)
Muted	Sets if the audio output should be muted
PlaybackRate	Sets the content playback speed
URL	Sets the location of the media file
Volume	Sets the volume of the audio output



TWebMediaCapture



Description

TWebMediaCapture is a non-visual component to capture data from a device microphone or camera. It allows to directly access the captured sound or video as binary data.

TWebMediaCapture is ideal to measure audio levels for example. Below is a list of the most important properties methods and events for TWebMediaCapture.

 <p>Designtime</p>	 <p>Runtime</p>
--	---

Properties for TWebMediaCapture

Camera	Sets the TWebCamera component from where video capture will done
Capture	Specifies what source to capture: mctBoth: both video and audio mctAudio: capture only audio mctVideo: capture only video
FFTSize	Sets the size (in sample points) of the data used for an FFT (Fast Fourier Transform) for audio level calculation.
RecordingMode	Selects between manual or automatic recording mode mrmManual: record after programmatically start & stop mrmAutomatic: start recording automatically after a critical audio level is reached

Sensitivity	Sets if the audio level sensitivity that triggers an automatic recording
SmoothTimeConstant	Constant used in calculation of the audio level over time

Methods for TWebMediaCapture

Start	Start the media recording
Stop	Stop the media recording

Events for TWebMediaCapture

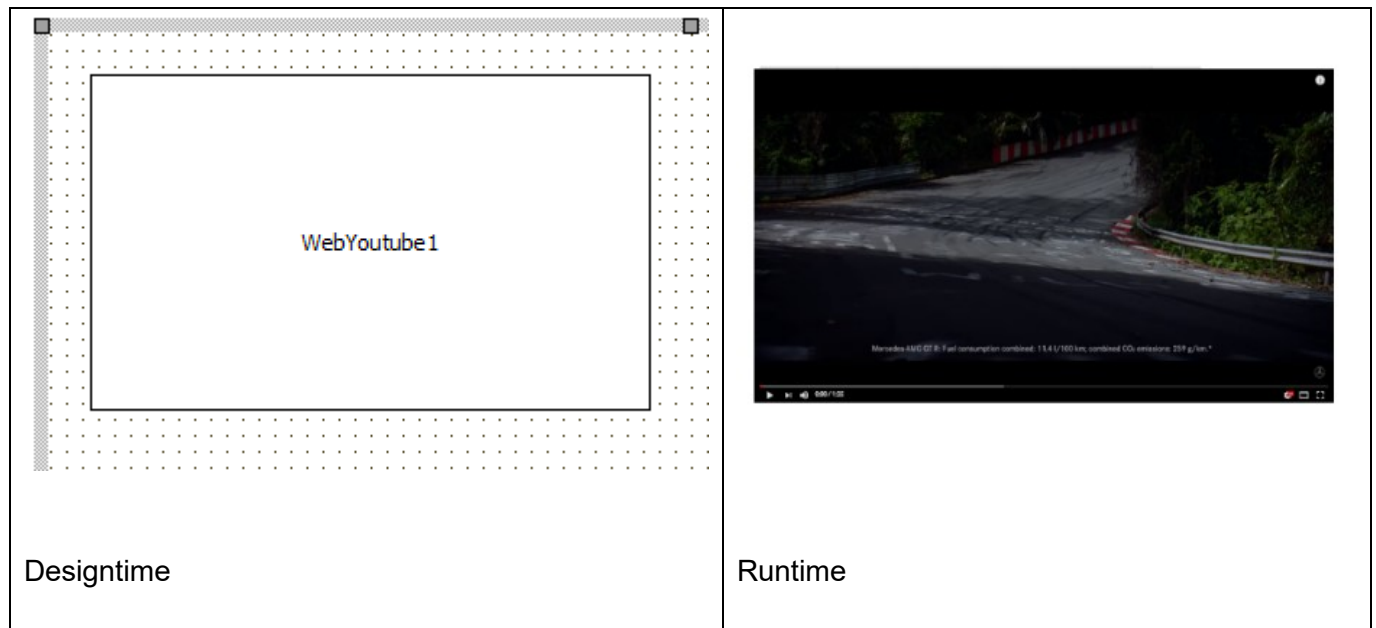
OnStartCapture	Event triggered when media capture has started
OnStopCapture	Event triggered when media capture has stopped returning the captured media data as binary data or an encoded string

TWebYoutube



Description

Below is a list of the most important properties methods and events for TWebYoutube.



HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<IFRAME ID="UniqueID"></IFRAME>
ElementID	UniqueID

Properties for TWebYoutube

AllowFullScreen	When true, the button to show the video in full screen is displayed
-----------------	---

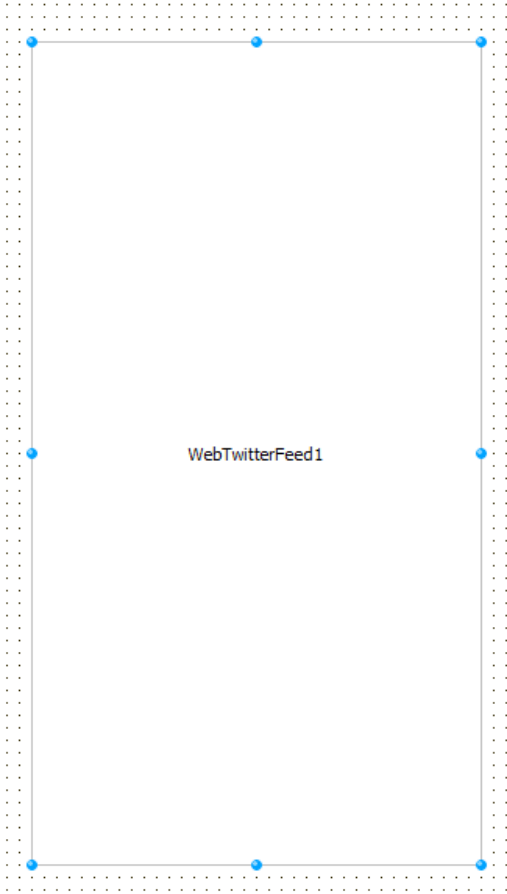
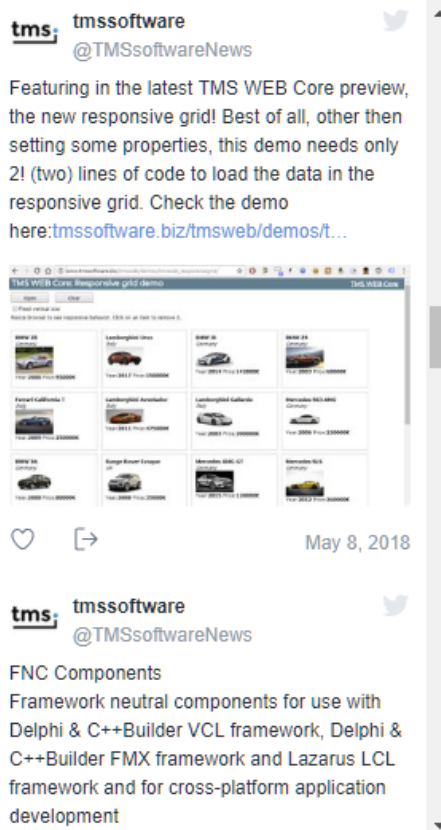
AutoPlay	When true, the video starts playing as soon as the page opens
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
VideoID	Sets the Youtube ID of the video

TWebTwitterFeed



Description

Below is a list of the most important properties methods and events for TWebTwitterFeed. TWebTwitterFeed is an easy way to display a Twitter feed in a page. The Twitter feed displays as soon as the Feed (Twitter ID) is set.

 <p>WebTwitterFeed1</p>	
Designtime	Runtime

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebTwitterFeed

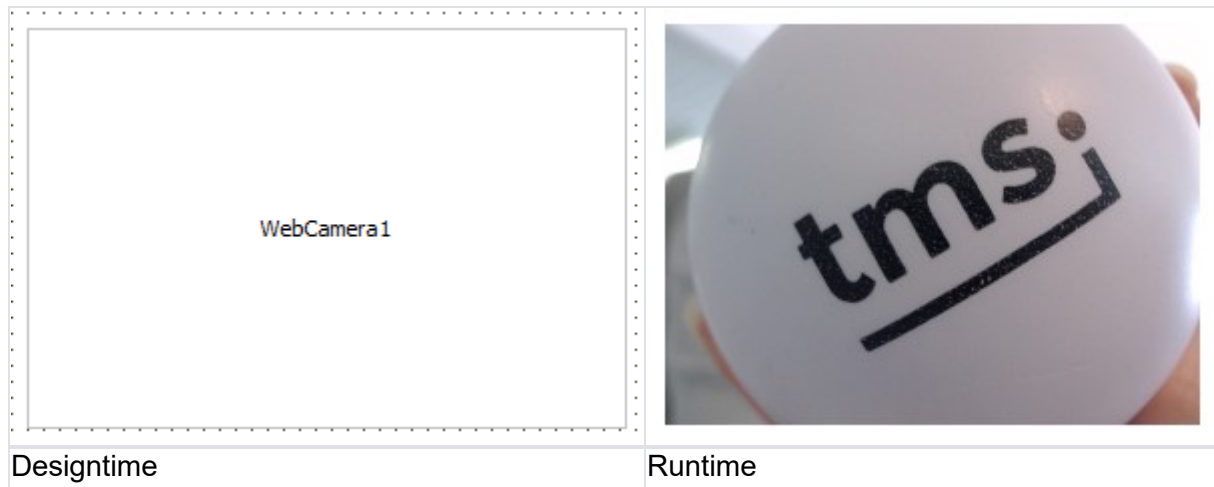
ElementClassName	Optionally sets the CSS classname for the label when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new label is created but the Delphi class is connected with the existing HTML element in the form HTML file
Feed	Sets the id of the Twitter feed to display
FeedLinkText	Sets additional text displayed together with the feed items

TWebCamera

Description

Below is a list of the most important properties methods and events for the TWebCamera. TWebCamera is using the [MediaDevices.getUserMedia\(\)](#) API. Because of this, two mayor limitations are:

- The TWebCamera won't work in any browser that does not support the getUserMedia API.
- It is not yet supported in iOS PWA.



Selecting a device

The initialization of the available camera devices is an async process. The setup requires a few steps but with the provided properties and events you can create a list for the user to pick their preferred camera to use.

Suppose a TWebCamera is already available on the form. Set the CameraType property to ctSelected. In this example we will use a TWebComboBox to create a list of devices.

In the OnCameraDevicesInitialized event we can fill the TWebComboBox:

[view plain text](#)

```
1. procedure TForm1.WebCamera1CameraDevicesInitialized(Sender: TObject);
2. var
3.     I: Integer;
4.     d: TCameraDevice;
5. begin
6.     for I := 0 to WebCamera1.CameraDevices.Count - 1 do
7.         WebComboBox1.Items.Add(WebCamera1.CameraDevices.Items[I].Name);
8.
9.     //If you want to select the first available device in the TWebComboBox, then:
10.    if WebComboBox1.Items.Count <> 0 then
11.        begin
12.            WebComboBox1.ItemIndex := 0;
13.
14.            //If you want to start the camera stream immediately
15.            //with the first selected device, then:
16.            d := WebCamera1.CameraDevices.GetDeviceByName(WebComboBox1.Items[0]);
17.            WebCamera1.SetSelectedCameraDevice(d);
18.            WebCamera1.Start;
19.        end;
20. end;
```

And to handle the selection of the device from the user, we can use the OnChange event of the TWebComboBox:

[view plain text](#)

```
1. procedure TForm1.WebComboBox1Change(Sender: TObject);
2. var
3.     d: TCameraDevice;
4. begin
5.     d := WebCamera1.CameraDevices.GetDeviceByName(WebComboBox1.Items[WebComboBox1.ItemIndex]);
6.     WebCamera1.SetSelectedCameraDevice(d);
7. end;
```

Now you can call WebCamera1.Start when the camera stream needs to be started.

Starting the camera stream automatically

If the component would start the camera streaming itself if the selected devices has changed, then it might lead to undesirable behavior in some applications. Therefore, this is something the developer have to take care of themselves. If you want to start the camera as soon as the selected device has changed, then you can do so by using the OnSwitchCamera event:

[view plain text](#)

```
1. procedure TForm1.WebCamera1SwitchCamera(Sender: TObject;
2.     ACamera: TCameraDevice);
3. begin
4.     WebCamera1.Start;
5. end;
```

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<VIDEO ID="UniqueID"></VIDEO>
ElementID	UniqueID

Properties for TwebCamera

Property	Description
BrowserSupportedConstraints: TStringList	Public property for settings for camera constraints See: https://developer.mozilla.org/en-US/docs/Web/API/Media_Streams_API/Constraints

Property	Description
CameraDevices: TCameraDevices	A read-only property to retrieve a collection of camera devices that are available.
CameraType: TCameraType	Set or retrieve the camera type. Available values are: ctFront, ctRear, ctSelected. In case of ctFront and ctRear the component will try to use the preferred camera. ctSelected is used in combination with the CameraType property, where a selected camera must be set based on the available devices.
Paused: Boolean	A read-only property to retrieve if the camera is in a paused state.
SnapShotAsBase64: string	A read-only property to retrieve a snapshot from the camera as a Base64 encoded string.
SnapShotAsImageData: TJSImageData	A read-only property to retrieve a snapshot from the camera as a TJSImageData.

Methods for TWebCamera

Property	Description
Pause	Method to pause the camera stream.
Resume	Method to return to the paused camera stream.
SetSelectedCameraDevice(aDevice: TCameraDevice)	Method to set the selected camera device.
Start	Method to start the camera stream.
Stop	Method to stop the camera stream completely.

Events for TWebCamera

Property	Description
OnBeforeStart	Event triggered before the camera starts recording
OnCameraDevicesInitialized	Event triggered when the camera devices are initialized and available.
OnCameraPause	Event triggered when the camera gets paused.
OnCameraResume	Event triggered when the camera resumes.
OnCameraStreamPlay	Event triggered when the camera stream starts playing.
OnCameraStop	Event triggered when the camera stream stops.
OnClick	Event triggered when the control is clicked.

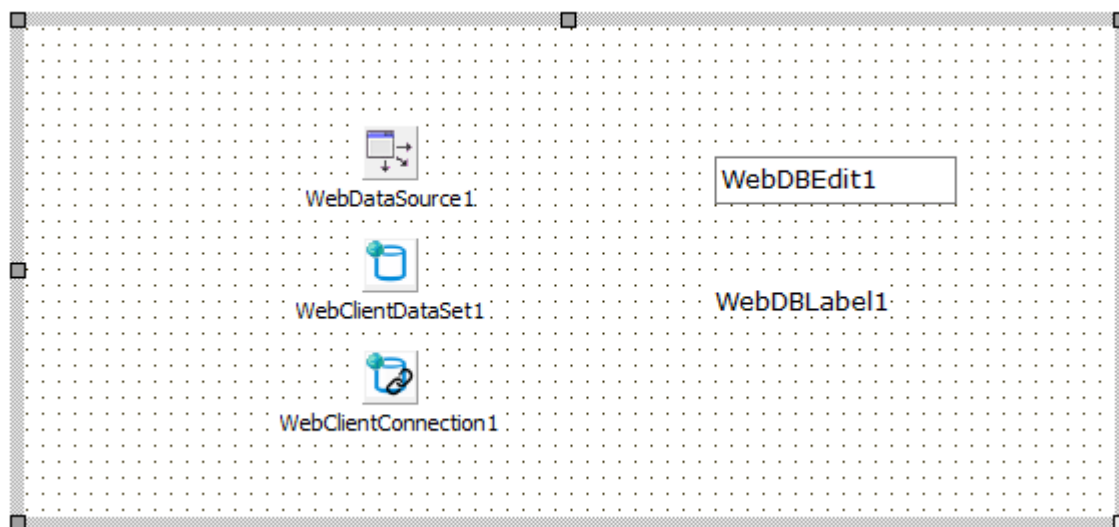
Property	Description
OnBeforeStart	Event triggered before the camera starts recording
OnDbClick	Event triggered when the control is double clicked.
OnMouseDown	Event triggered when the mouse is down on the control.
OnMouseEnter	Event triggered when the mouse enters the control.
OnMouseLeave	Event triggered when the mouse leaves the control.
OnMouseUp	Event triggered when the mouse goes up on the control.
OnMouseMove	Event triggered when the mouse moves on the control.
OnSwitchCamera	Event triggered when the selected camera device changes.

DB-aware components

TMS WEB Core offers the concept of a dataset and datasource. Via a dataset and a datasource, UI controls can be directly connected to a dataset, avoiding to write any code to show data and update data.

Databinding works similar as in VCL application. A DB-aware control has a DataSource property that is connected to a non-visual datasource component (TWebDataSource). The TWebDataSource is in turn connected to a dataset, for example the TWebClientDataSet. Other than the DataSource property, the DB-aware control uses the FieldName property to select the DB field with which to connect the DB-aware control.

The non-visual datasource and dataset components can be placed directly on the form, or even better, on a TWebDataModule.



TWebDataSource



Description

TWebDataSource provides an interface between a dataset component and data-aware controls on a form. Use TWebDataSource to provide a conduit between a dataset and data-aware controls on a form that enable display, navigation, and editing of the data underlying the dataset. All datasets must be associated with a data source component if their data is to be displayed and manipulated in data-aware controls. Similarly, each data-aware control needs to

be associated with a data source component in order for the control to receive and manipulate data.

Properties for TWebDataSource

AutoEdit	Determines if a data source component automatically calls a dataset's Edit method when a data-aware control associated with the data source receives focus.
DataSet	Specifies the dataset for which the data source component serves as a conduit to data-aware controls or other datasets.
Enabled	Determines if the data-aware controls associated with the data source component display data.

TWebClientDataSet



Description

TWebClientDataSet is the class for an in browser memory dataset. Client datasets can work with data retrieved from a REST request or by directly assigning JSON arrays. They cache that data in memory, maintain a record of any changes in a change log, and apply cached updates at a later point back to the source of the data.

Properties for TWebClientDataSet

Active	Specifies whether or not a dataset is open.
Connection	Sets the TWebClientConnection component that can take care of performing the REST requests to load the data in the TWebClientDataSet.
DataSource	Represents the data source of another dataset that supplies values to the dataset.
Fields	Use Fields to access field components. If fields are generated dynamically at runtime, the order of field components in Fields

	corresponds directly to the order of columns in the table or tables underlying a dataset. If a dataset uses persistent fields, then the order of field components corresponds to the ordering of fields specified in the Fields editor at design time.
FieldDefs	Points to the list of field definitions for the dataset.
Params	Use Params to specify parameter values that the provider should pass to a source dataset
RecNo	Indicates the active record in the dataset.
RecordCount	Returns the number of records in the dataset
Rows: TJSArray	JSON array property allow to set the dataset data from a JSON array

Methods for TWebClientDataSet

ApplyUpdates	Sends all updated, inserted, and deleted records from the client dataset to the provider for writing to the database.
Cancel	Cancels unposted changes to the current record.
ClearFields	Removes all fields from the fields collection
Close	Closes the dataset. Equivalent to setting Active = false
Delete	Deletes the active record and positions the dataset on the next record.
Edit	Sets the dataset in edit mode
EmptyDataSet	Removes all data (records) from the dataset
First	Moves to the first record in the dataset.
Insert	Puts the dataset in insert state
Last	Moves to the last record in the dataset.
Next	Moves to the next record in the dataset.
Open	Opens the dataset. Equivalent to setting Active = true
Post	Writes a modified record to the Data property or the change log.
Prior	Moves to the previous record in the dataset.

Events for TWebClientDataSet

AfterCancel	Event triggered after a cancel operation on the dataset
AfterClose	Event triggered after a dataset close
AfterDelete	Event triggered after a delete operation on the dataset
AfterEdit	Event triggered after the dataset was set in edit mode
AfterInsert	Event triggered after the dataset was set in insert mode
AfterOpen	Event triggered after a dataset open
AfterPost	Event triggered after a post operation on the dataset
AfterScroll	Event triggered after a scroll
BeforeCancel	Event triggered just before a cancel operation will be performed on the dataset
BeforeClose	Event triggered just before the dataset will be effectively closed
BeforeDelete	Event triggered just before a delete operation will be performed on the dataset
BeforeEdit	Event triggered just before the dataset is set into edit mode
BeforeInsert	Event triggered just before an insert operation will be performed on the dataset
BeforeOpen	Event triggered just before the dataset will be effectively opened
BeforePost	Event triggered just before a post operation will be performed on the dataset
BeforeScroll	Event triggered just before a scroll will happen in the dataset
OnCalcFields	Occurs when an application recalculates calculated fields.
OnDeleteError	Occurs when an application attempts to delete a record and an exception is raised.
OnEditError	Occurs when an application attempts to modify or insert a record and an exception is raised.
OnFilterRecord	Occurs each time a different record in the dataset becomes the active record and filtering is enabled.

OnNewRecord	Occurs when an application inserts or appends a new dataset record.
OnPostError	Occurs when an application attempts to modify or insert a record and an exception is raised.
OnUpdateRecord	Occurs when cached updates are applied to a record.

TWebClientConnection



Description

TWebClientConnection is a non-visual component that can take of the loading of TWebClientDataSet data via a HTTP request returning a JSON array.

Properties for TWebClientConnection

Active	Property to set the connection to active. Setting Active = true means the TWebClientConnection will try to fetch the data from the URL that is set with the URI property
AutoOpenDataSet	When true, the dataset using the TWebClientConnection will be automatically set to Active = true after the JSON array response of the HTTP request is loaded
DataNode	Sets an optional JSON node name under which the JSON array of data can be found.
Headers	Can contain optional HTML headers to be sent to the server when making the HTTP(s) request to retrieve the data
Password	Sets the password to be used in case the HTTP(s) request needs authentication
URI	Sets the URL

User	Sets the user name to be used in case the HTTP(s) request needs authentication
------	--

Events for TWebClientConnection

AfterConnect	Event triggered after the connection was successful
BeforeConnect	Event triggered before the HTTP(s) request will be performed
OnConnectError	Event triggered when the HTTP(s) request was unsuccessful
OnDataReceived	Event triggered when data from the server was returned.

TWebDBLabel



Description

This is a DB-aware label. The label connects typically to a DB string field and shows the content of the DB string field as label on the form.

The TWebDBLabel is connected via DataSource and DataField properties to a dataset.

TWebDBEdit



Description

This is a DB-aware edit control. The edit control connects typically to a DB string field and allows to edit the content of the DB string field via an edit control on the form.

The TWebDBEdit is connected via DataSource and DataField properties to a dataset.

TWebDBEditAutoComplete



Description

This is a DB-aware edit control with auto completion based on a preset list of strings. The edit control connects typically to a DB string field and allows to edit the content of the DB string field via an edit control on the form.

The TWebDBEditAutoComplete is connected via DataSource and DataField properties to a dataset.

TWebDBCheckBox



Description

This is a DB-aware checkbox control. The checkbox control connects typically to a DB boolean field and allows to edit the content of the DB Boolean field via a checkbox control on the form.

The TWebDBCheckBox is connected via DataSource and DataField properties to a dataset.

TWebDBSpinEdit



Description

This is a DB-aware spin edit control. The spin edit control connects typically to a DB numeric field and allows to edit the content of the DB numeric field via a spin edit control on the form.

The TWebDBSpinEdit is connected via DataSource and DataField properties to a dataset.

TWebDBMaskEdit



Description

This is a DB-aware mask edit control. The mask edit control connects typically to a DB field (numeric / date / text) and allows to edit the content of the DB field via a mask edit control on the form.

The TWebDBMaskEdit is connected via DataSource and DataField properties to a dataset.

TWebDBComboBox



Description

This is a DB-aware combobox control. The combobox control connects typically to a DB string field and allows to edit the content of the DB string field via an edit control on the form.

The TWebDBComboBox is connected via DataSource and DataField properties to a dataset.

TWebDBLookupComboBox



Description

This is a DB-aware lookup combobox control. The combobox control connects typically to a DB string field and allows to edit the content of the DB string field via an edit control on the form. The value stored in the DB is the Value part of the Value/DisplayText pair while the text displayed in the combobox maps to the DisplayText value.

The TWebDBLookupComboBox is connected via DataSource and DataField properties to a dataset.

TWebDBMemo



Description

This is a DB-aware memo control. The memo control connects typically to a DB text blob field and allows to edit the content of the DB text blob field via a memo control on the form.

The TWebDBMemo is connected via DataSource and DataField properties to a dataset.

TWebDBDateTimePicker



Description

This is a DB-aware date or time picker control. The date or time picker control connects typically to a DB date or time field and allows to edit the content of the DB date or time field via a date or time picker control on the form.

The TWebDBDatePicker is connected via DataSource and DataField properties to a dataset.

TWebDBRadioGroup



Description

This is a DB-aware radiogroup control. The radiogroup control connects typically to a DB integer field and allows to edit the content of the DB integer field via a group box control on the form.

The TWebDBRadioGroup is connected via DataSource and DataField properties to a dataset.

TWebDBLinkLabel



Description

This is a DB-aware link label control. The link label control connects typically to a DB string field and allows to show the content of the DB string field via a label with link on the form.

The TWebDBLinkLabel is connected via DataSource and DataField properties to a dataset.

TWebDBImageControl



Description

This is a DB-aware image control. The image control connects typically to a DB string field and allows to show the content of the DB string field as an image referring to the URL in the DB string field value.

The TWebDBImageControl is connected via DataSource and DataField properties to a dataset.

For setting generating the proper image URL from the DB field value, two additional capabilities are offered.

BaseURL	Sets the optional URL prefix. In case the DB field only contains the image filename, BaseURL can be set to the full HTTP(S) URL specifier
OnSetURL	This event is triggered with a var parameter AURL that can be used to transform the DB field value to the required full HTTP(S) URL

TWebDBTableControl



Description

This is a DB-aware table control. A table control column connects typically to a DB field and allows to show the content of the DB field in a column of the table.

The column in the TWebDBTableControl.Columns collection has following properties:

DataField	Sets the DB field that should be displayed in the column
DataType	Defines whether the DB field connected to the column should be displayed as text, an image or a hyperlink
Title	Sets the column header text

TWebDBResponsiveGrid



Description

This is a DB-aware responsive list control. A responsive list control column connects typically to a DB field and allows to show the content of the DB fields in a list item via a template.

The template configures the HTML to be displayed in a responsive list item. The template is set via TWebDBResponsiveGrid.Options.ItemTemplate.

To include a DB field value in the item, specify in the template the DB field as:
(%FIELDNAME%)

Example:

When connecting the FishFact JSON dataset to the responsive list and setting the template in the following way:

```
TWebDBResponsiveGrid.Options.ItemTemplate :=
  '<strong>(%_Common_Name%)</strong><br>(%_Species_Name%)<br><IMG
width="96px" src="(%_Graphic%) ">';
```

The result is that from the dataset, the `_Common_Name`, `_Species_Name` field are shown and the `_Graphic` field image URL is used to show the image with a width of 96 pixels:



TWebDBGrid



Description

This is a DB-aware grid. A grid column connects typically to a DB field and allows to show the content of the DB field in a column of the grid.

It inherits all properties, methods & events of the non DB-aware TWebStringGrid.

The column in the TWebDBGrid.Columns collection has following properties:

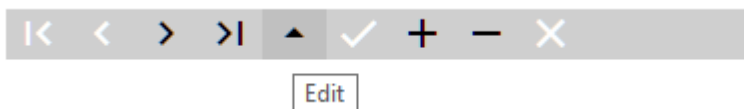
DataField	Sets the DB field that should be displayed in the column
DataType	Defines whether the DB field connected to the column should be displayed as text, an image or a hyperlink
ElementClassName	Sets an optional CSS class name for the cells of the column
Title	Sets the column header text
TitleElementClassName	Sets an optional CSS class name for the header cell of the column
Width	Sets the width of the column (in pixels)

TWebDBNavigator



Description

This is a DB-aware navigator, allowing to scroll in the connected dataset and perform operations as Edit, Post, Cancel on the dataset.



To use the TWebDBNavigator, drop it on the form and connect the datasource.

With the property VisibleButtons, set what buttons in the navigator need to be visible. The VisibleButtons property is a set property consisting of the following possible values:

nbFirst, nbPrior, nbNext, nbLast, nbInsert, nbDelete, nbEdit, nbPost, nbCancel

To customize the hint setting for each of the controls in the navigator, the TWebDBNavigator.Hints property can be used.

Non-visual components and classes

TWebTimer



TWebTimer is the direct equivalent of a VCL TTimer. It features an interval property with which the interval between two subsequent OnTimer events can be set in milliseconds. With the Enabled property the timer can be stopped or started. When the timer is enabled, it triggers the OnItem event every 'interval' milliseconds

TWebClipboard



TWebClipboard is a non-visual control that manages paste from the clipboard at window level in the browser. When the user performs paste either from the browser menu or via the keyboard shortcut Ctrl-V, the TWebClipboard.OnTextData or TWebClipboard.OnImageData is triggered. When the user pasted text, OnTextData is triggered returning the text. When the user pasted an image, the event OnImageData is triggered returning the image as base64 encoded data URL.

The TWebClipboard component also allows to programmatically put text on the clipboard. This can be done via:

```
TWebClipboard.CopyToClipboard(const AValue: string);
```

TWebBluetooth



TWebBluetooth is a component wrapping the web Bluetooth API for communicating from the browser with Bluetooth devices.

Bluetooth communications are setup via a Bluetooth device using a Bluetooth service that can read/write values via Bluetooth Characteristics.

Therefore, the TWebBluetooth class permits to make a connection to a device that can be accessed via the class TBluetoothDevice. Via the TBluetoothDevice, access to a service, made available via the TBluetoothService class, can be obtained. Values can be read or written using a characteristic, exposed via the class TBluetoothCharacteristic.

TWebBluetooth class

Public methods

function HasBluetooth: boolean	Returning whether the browser supports or does not support Bluetooth
function GetDevice: boolean	Try to establish a connection to a device and return an instance
function GetDevice(proc: TBTRefProc): boolean;	Function with anonymous method to establish a connection to a device
property Device: TBluetoothDevice	Access to the last connected device object

Published properties / events

DeviceName	Sets the name of the Bluetooth device when connection to only a specific device is wanted. Leave empty when a connection to just any Bluetooth device can be made
FilterService	Stringlist holding one or more services a Bluetooth device must offer before a connection to it can be made
OnDeviceObject	Event triggered when a device is connected, returning the device object
OnDeviceError	Event triggered when an error in the communication with the device is encountered.

TWebBluetoothDevice class

Public methods

function HasBluetooth: boolean	Returning whether the browser supports or does not support Bluetooth
function GetService: boolean	Try to obtain a service object reference from the device. The service is returned via the OnService event
function GetService(proc: TBTRefProc): boolean;	Function with anonymous method to get a service object
function GetServices;	Try to query for all services the device exposes. Services are returned via the

	OnServices event.
procedure Connect	Make a connection to the device. When successful, the OnConnect event is triggered.
procedure Connect(proc: TBTRefProc)	Make a connection to the device using an anonymous
procedure DisconnectDevice	Disconnect from the device
procedure ReConnectDevice	Try to establish a new connection to the device
function Connected: boolean	Returns true when a connection to the device could be established
Property Service: TBluetoothService	Reference to the last retrieved service object

Published properties / events

OnConnect	Event triggered when a connection to the device could be established
OnDisconnect	Event triggered upon disconnect
OnService	Event triggered when a device service is retrieved
OnServices	Event triggered when the list of supported services by the device is returned

Example:

This code snippet shows how a service can be obtained from a device

```
WebBluetooth.Device.GetService(tempervice,
  procedure(AService: TBluetoothService)
  begin
    myservice := AService;
  end
);
```

TWebBluetoothService class

Public methods

procedure GetCharacteristic(uuid: string);	Retrieve a characteristic with ID UUID from a service. When available, the characteristic is returned via the OnCharacteristic event.
procedure GetCharacteristic(uuid: string; proc:	Retrieve a characteristic with ID UUID from a

TBTCharacteristicProc);	service. When available, the characteristic is returned via an anonymous method.
procedure GetCharacteristics;	Query all characteristics offered by the service. The list of available services is returned via the event OnCharacteristics

Published properties / events

UUID	The UUID of the service
OnCharacteristic	Event triggered when a characteristic is requested
OnCharacteristics	Event triggered when the list of characteristics is requested

Example:

This code snippet shows how a characteristic is retrieved from a service:

```

AService.GetCharacteristic(tempcharval,
    procedure (AChar: TBluetoothCharacteristic)
    begin
        btchartempvalue := AChar;
    end
);

```

TWebBluetoothCharacteristic class

Public methods

procedure StartNotify	Method to start the notify mechanism. When started, the Bluetooth device will send a message (and trigger the OnNotifyXXX) event when a value of a characteristic changes.
procedure StopNotify	Stops the notify mechanism of the Bluetooth device
procedure ReadXXX	Read a value from the Bluetooth characteristic. The default Read performs a read on an integer value. For other types, XXX stands for different types:

	<p>Byte Int SmallInt Single Double String Array</p> <p>The result of the read is returned via the matching OnReadXXX event.</p>
<p>procedure ReadXXX(proc: TBTReadValueProc)</p>	<p>Read a value from the Bluetooth characteristic and the result is returned via an anonymous method.</p> <p>XXX stands for different types:</p> <p>Byte Int SmallInt Single Double String Array</p>
<p>Procedure WriteXXX()</p>	<p>Write a value to a Bluetooth characteristic.</p> <p>XXX stands for different types:</p> <p>Byte Int SmallInt Single Double String Array</p>

Published properties / events

UUID	The UUID of the characteristic
OnReadXXX	<p>Event triggered returning the result of a read operation.</p> <p>There are different variants of the read event for different data types</p>
OnNotifyXXX	<p>Event triggered when a new characteristic value is available when the notification mechanism was enabled.</p> <p>There are different variants of the notify event for different data types</p>

Example

This example shows how to read an

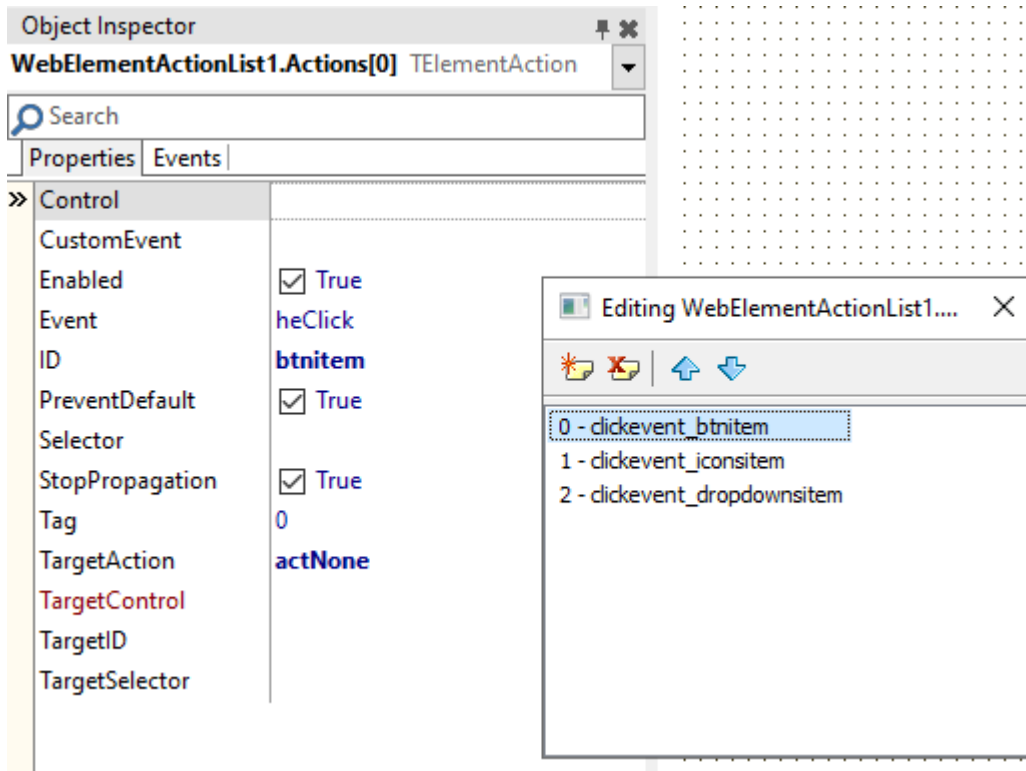
```
btchartempvalue.Read(  
  procedure(AValue: integer)  
  begin  
    ReadCharacteristic(Self, AValue);  
  end  
);
```

TWebElementActionList



Description

A TWebElementActionList should not be confused with a Delphi TActionList. The purpose of a TWebElementActionList is to easily hookup events to HTML elements typically available in a form template but not limited to these. Binding to event handlers of HTML elements is based on the HTML element ID, a query selector or a Pascal control class. TWebElementActionList is a list of TWebElementAction items where each such item represents the bridge between HTML elements and the action perform when an event occurs on these HTML elements.



If there is for example a HTML BUTTON element in the HTML template with ID “btn”, it is possible to define the UI logic that should be executed when the button is clicked by adding a TWebElementAction item, setting the ID for the button, the event to heClick and then write the OnExecute event for the TWebElementAction. This OnExecute event will be triggered when the button is clicked.

Note that multiple TWebElementAction items can be bound for different events to the same HTML element or elements.

Properties for TWebElementActionList

Actions	Collection of actions that specify for what HTML element event an action OnExecute or actionlist OnExecute needs to be triggered
---------	--

Events for TWebElementActionList

OnExecute	Event triggered when the HTML element or elements event specified by the TWebElementAction happens. The event passes
-----------	--

	the action, the HTML element triggering the action and the JavaScript event object.
OnUpdate	Event triggered for each HTML element(s) set by TargetControl or TargetID or TargetSelector. The event passes the action, the HTML element triggering the action, the JavaScript event object and the HTML element that is the target of the action.

Properties for TWebElementAction

Control	Sets the Pascal control to bind a specific event to
CustomEvent	Sets the event type as string to bind to in case the event type is not in the list of standard events.
Enabled	When true, the OnExecute event will be triggered when the bound event on the element is happening.
Event	Specifies what specific HTML event will trigger the action OnExecute event. The predefined event types are: heClick: click on the HTML element heDbClick: double-click on the HTML element heKeyPress: keypress on HTML element heKeyDown: key down on HTML element heKeyUp: key up on HTML element heMouseDown: mouse down on HTML element heMouseMove: mouse move on HTML element heMouseUp: mouse up on HTML element heMouseEnter: mouse enter on HTML element heMouseLeave: mouse leave on HTML element heBlur: focus leave from HTML element heFocus: focus enter on HTML element heChange: value change on HTML element heSelect: selection on OPTION HTML element heInvalid: invalid input on HTML element heCustom: custom event (set by CustomEvent property) heNone: no element event is bound
ID	Sets the HTML element ID for the element to bind the action to
Name	Name of the item instance
PreventDefault	When true, the default HTML event handler for the element will not be executed. For example, a key event will not have effect on the element.

Selector	Sets the query selector for possibly multiple HTML elements to bind with the TWebElementAction. For example, specifying 'INPUT' will select all HTML INPUT elements in the document to bind the action to. How selectors can be used to do sophisticated selection of HTML elements can be found here: https://www.w3schools.com/cssref/css_selectors.asp
StopPropagation	When true, the event on the HTML element doesn't propagate to its container element. For example, a mouse down event is propagated to the container element when not handled by the first HTML element that gets it.
Tag	Integer value
TargetAction	Action to perform when the event is happening on the target elements. actNone: no action performed on target elements actSetHidden: set target elements display attribute as hidden actRemoveHidden: remove target elements display attribute as hidden actToggleHidden: toggle target elements display attribute actSetReadOnly: set target elements readonly attribute actRemoveReadOnly: remove target elements readonly attribute actToggleReadOnly: toggle target elements readonly attribute actSetDisabled: set target elements disabled attribute actRemoveDisabled: remove target elements disabled attribute actToggleDisabled: toggle target elements disabled attribute actClear: clears the value of the target elements
TargetControl	The Pascal control affected by the TWebElementAction when its event occurs
TargetID	ID of the HTML event affected by the TWebElementAction when its event occurs
TargetSelector	Sets the query selector for possibly multiple HTML elements that an action will have effect on.

Methods for TWebElementAction

Bind	Binds the action class to the HTML element(s) specified by Control or ID or Selector. The TWebElementActionList will already implicitly perform binding upon creation of the class. Bind only needs to be called in case a TWebElementAction is created at runtime
UnBind	Unbinds the action class from the HTML element(s) specified by

	Control and/or ID and/or Selector. This normally implicitly happens when the TWebElementActionList is destroyed
--	---

Events for TWebElementAction

OnExecute	Event triggered when the HTML element or elements event specified by the TWebElementAction happens
OnUpdate	Event triggered for each HTML element(s) set by TargetControl or TargetID or TargetSelector

Example

For a HTML template that contains an entry form, we can easily add a TWebElementAction to clear the entered fields when the Clear button is clicked. The Clear button has the ID “btnclear”, so a new TWebElementAction object is added to the list and the event is set to heClick. As the button click should result in clearing HTML input elements, set TWebElementAction.TargetAction to actClear. Finally set TargetSelector to ‘input.forminput, textarea.forminput, select.forminput’ to get all elements in a form, i.e. that have class set to forminput.

Code

```
var
  wa: TWebElementAction;

begin
  wa := TWebElementActionList.Actions.Add;
  wa.ID := 'btnclear';
  wa.Event := heClick;
  wa.TargetAction := actClear;
  wa.TargetSelector := 'Input.forminput, textarea.forminput,
select.forminput';
end;
```

TWebLocalStorage

TWebLocalStorage is a class that can be used to access local browser storage. The storage is handled by the browser and coupled to the specific URL of the web application. From another URL, this local storage is not accessible. The TWebLocalStorage can be considered as a key/string value pair storage. With the class TWebLocalStorage it is easy to use.

Example:

```
var
    LLocalStorage: TWebLocalStorage

    LLocalStorage := TWebLocalStorage.Create;
    LLocalStorage.Values['mykeyname'] := 'myvalue';
    LLocalStorage.Free;
```

Or alternatively, you can also use the static method that reduces this code to:

```
TWebLocalStorage.Values['mykeyname'] := 'myvalue';
```

When the same URL is visited by the browser, the values stored from the last session can be retrieved.

TWebSessionStorage

TWebSessionStorage is similar to TWebLocalStorage except that values are only persisted in the browser for the lifetime of the session. Just like for TWebLocalStorage, it consists of a key/value pair storage.

Example:

```
var
    LSessionStorage: TWebSessionStorage

    LSessionStorage:= TWebSessionStorage.Create;
    LSessionStorage.Keys['mykeyname'] := 'myvalue';
    LSessionStorage.Free;
```

TWebURLValidator



TWebURLValidator is a non-visual component that allows to perform a check whether an URL exists and works or not. Set the URL to test via TWebURLValidator.URL and call the Validate method. This will trigger the OnValidated event where the IsValid parameter will return whether the URL is valid or not.

Example:

```
procedure TForm1.WebFormCreate(Sender: TObject);
begin
    WebURLValidator1.URL := 'http://myurltotest.com';
    WebURLValidator1.Validate;
end;

procedure TForm1.WebURLValidator1Validated(Sender: TObject; IsValid:
Boolean);
begin
    if IsValid then
        ShowMessage('The URL ' + WebURLValidator1.URL + ' works!');
end;
```

Properties for TWebURLValidator

URL	Sets the URL to check if it exists
-----	------------------------------------

Events for TWebURLValidator

OnValidated	Event triggered when the URL has been validated and returning whether it was a valid URL or not.
-------------	--

TWebGeoLocation



TWebGeoLocation wraps the browser capability to determine the geolocation of the device on which the browser runs. For privacy reasons, when an attempt to retrieve the geo location is performed, it will trigger a popup dialog requesting the authorization from the user to do so. With the method TWebGeoLocation.GetGeoLocation the request to get the geo location is started. When the geo location is retrieved, the OnGeoLocation event is triggered returning the longitude, latitude and altitude of the location.

```
procedure TForm1.WebGeoLocation1Geolocation(Sender: TObject; Lat, Lon,
    Alt: Double);
begin
    WebLabel1.Caption := Format('Device is at [%.4f:%.4f]', [Lon,Lat]);
end;
```

Note: use of TWebGeoLocation requires for privacy & security SSL (i.e. app needs to be hosted on a HTTPS enabled domain).

Methods for TWebGEOLocation

GetGeolocation	Start the asynchronous retrieval of the geolocation of the device where the browser is running.
----------------	---

Events for TWebGEOLocation

OnGeoLocation	Event triggered when the geolocation was retrieved. This returns via the event parameters the longitude, latitude and altitude.
---------------	---

TWebSocketClient



The TWebSocketClient is a non-visual component enabling to perform web socket communication with a websocket server.

Set the hostname and port of the websocket server via WebSocketClient.HostName and Port. Start connecting to the websocket server via calling the method WebSocketClient1.Connect. When a successful connection is made, the WebSocketClient.OnConnect is triggered. Call WebSocketClient.Disconnect to disconnect from the server. When a disconnect is called programmatically or for another reason the connection to the websocket server is lost, the OnDisconnect event is triggered.

Sending & retrieving data

Data is sent as a string and retrieved as JavaScript object.

To send a command call:

```
WebSocketClient.Send(AMessage: string); overload;
```

```
WebSocketClient.Send(ABuffer: TJSArrayBuffer); overload;
```

When data is received from the websocket server, the event OnDataReceived is triggered. This returns the data as JavaScript. When the data is a string, the JavaScript object can be converted easily to a string by calling TJLObject.toString;

```
procedure TForm1.WebSocketClient1DataReceived(Sender: TObject; Origin:
string;
  Data: TJLObject);
begin
  WebListBox1.Items.Add(Data.toString);
end;
```

Properties for TWebSocketClient

HostName	Sets the name of the web socket server
PathName	Sets the (optional) path name for the socket server

Port	TCP/IP port to use for the web socket communication
------	---

Methods for TWebSocketClient

Send(AMessage: string);	Sends data to the socket server as string
Send(ABuffer: TJSArrayBuffer)	Sends data to the socket server as JavaScript byte array buffer

Events for TWebSocketClient

OnConnect	Event triggered when the web socket client could successfully connect to the server
OnDataReceived	Event triggered when data is received from the web socket server
OnDisconnect	Event triggered when the web socket client was disconnected from the web socket server

TWebHttpRequest

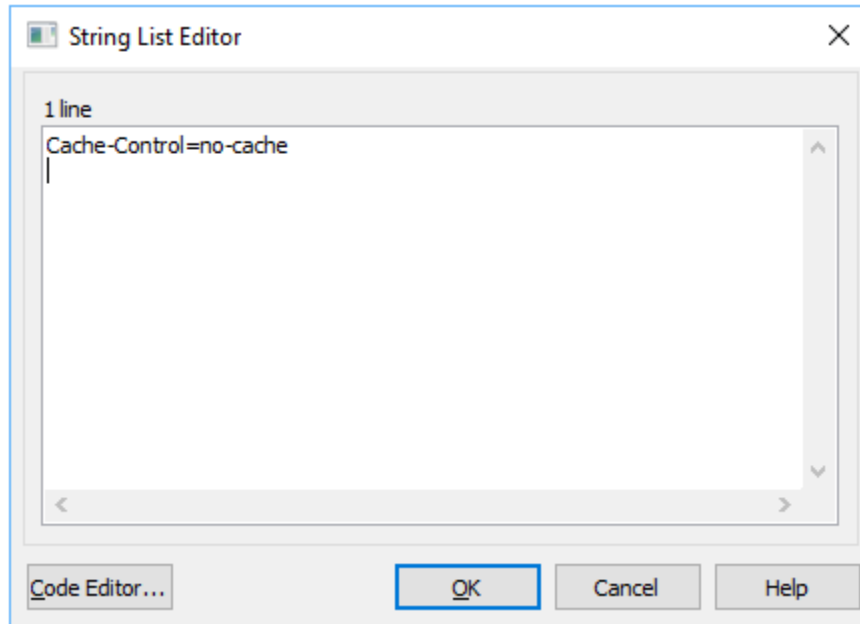


The TWebHttpRequest is a component to perform HTTP(s) requests to a server. The HTTP requests command can be:

httpCUSTOM : a custom HTTP command set with WebHttpRequest.CustomCommand
httpDELETE : a HTTP DELETE command
httpGET : a HTTP GET command (default)
httpHEAD : a HTTP HEAD command
httpPOST : a HTTP POST command
httpPUT : a HTTP PUT command

Optionally, HTTP request headers can be set. The HTTP request headers are set via WebHttpRequest.Headers. This is a value/pair list of HTTP options. Default, the option Cache-

Control is set to no-cache.



When a HTTP POST command is execute, the POST data can be set via the `WebHttpRequest.PostData` property.

By default `WebHttpRequest.Timeout` is zero, which means there is no time-out value. When wanting to set a time-out value, `WebHttpRequest.Timeout` sets the time-out in milliseconds.

Finally, the URL for performing the HTTP request is set via `WebHttpRequest.URL`: string;

When the HTTP request is successful, the `OnResponse` event is triggered. When it fails, the event `OnAbort` is triggered.

When the request is successful, the request response is returned as event parameter of the `OnResponse` event.

```
procedure TForm1.WebHttpRequest1Response(Sender: TObject; AResponse:
string);
begin
    ShowMessage('server response:' + AResponse);
end;
```

When the response comes as JSON, the JSON parser with a similar interface as the standard Delphi JSON parser can be used:

The following example shows how the response can be parsed as a JSON array:

```
procedure TForm1.WebHttpRequest1Response(Sender: TObject; AResponse:
string);
var
  JS: TJSON;
  JA: TJSONArray;
  JO: TJSONObject;
  i: integer;

begin
  JS := TJSON.Create;
  JA := TJSONArray(JS.Parse(AResponse));

  for i := 0 to JA.Count - 1 do
  begin
    JO := TJSONObject(JA.Items[i]);
    WebListBox1.Items.Add(JO.Get('prop'));
  end;
end;
```

An alternative way to handle the response is via an anonymous method. The signature of this anonymous method is declared as:

```
procedure(AResponse: string; ARequest: TJXMLHttpRequest);
```

The same example handled via an anonymous method as such becomes:

```
begin
  WebHttpRequest1.URL :=
'http://www.tmssoftware.biz/tmsweb/music.json';

  WebHttpRequest1.Execute(

    procedure(AResponse: string; AReq: TJXMLHttpRequest)
    var
      js: TJSON;
      ja: TJSONArray;
      jo: TJSONObject;
      i: integer;
    begin
      js := TJSON.Create;
```

```

try
  ja := TJSONArray(js.Parse(AResponse));

  ShowMessage('Retrieved items:' +inttostr(ja.Count));

  for i := 0 to ja.Count - 1 do
  begin
    jo := TJSONObject(ja.Items[i]);
    WebListBox1.Items.Add(jo.GetJSONValue('title'));
  end;
finally
  js.Free;
end;
end
);
end;

```

Properties for TWebHttpRequest

Command	Sets the HTTP command type to execute. This can be httpCUSTOM, httpGET, httpPOST, httpDELETE, httpHEAD, httpPUT
CustomCommand	Sets the custom HTTP command name
Headers	StringList holding optional header parameters to pass along with the HTTP command
Password	For authenticated HTTP requests, sets the password to be used
PostData	Sets the data to be posted along with a httpPOST command as string
TimeOut	Sets the timeout value (in milliseconds). This is the time after which the request should abort when not getting a response from the server
URL	URL for performing the HTTP request
User	For authenticated HTTP requests, sets the username to be used

Methods for TWebHttpRequest

Execute	Executes the HTTP request. An optional anonymous method can be used to catch the response
---------	---

Events for TWebHttpRequest

OnAbort	Event triggered when the HTTP request was aborted
OnError	Event triggered when an error occurred with the HTTP request
OnRequestResponse	Event triggered when a response for the HTTP request was received. This event returns both the response as string as well as the JavaScript response object
OnResponse	Event triggered when a response for the HTTP request was received. This event returns both the response as string
OnTimeOut	Event triggered when a timeout happened for the HTTP request

TWebCookies

TWebCookies is a collection class for managing cookies in your web application. It is defined in the unit WEBLib.Cookies. This is a collection of TWebCookie items. The TWebCookie item has following properties:

- property Name:string; gets or sets the cookie name/identifier
- property Value:string; gets or sets the cookie value
- property Expiry:TDateTime; gets or sets the cookie expiry date
- property Path: string; gets or sets the cookie path

The path parameter specifies a document location for the cookie, so it's assigned to a specific path, and sent to the server only if the path matches the current document location, or a parent:

To get the browser cookies for the application URL in the TWebCookies collection call TWebCookies.GetCookies.

For updating the cookies in the browser after making changes to the collection TWebCookie items, call TWebCookies.SetCookies.

Other TWebCookies collection methods:

procedure Delete(ACookie: TCookie);

Delete a cookie by instance

procedure Delete(const AName: string);

Delete a cookie by name

function Add(const AName, AValue: string; Expiry: TDateTime): TCookie;
function Add(const AName, AValue: string): TCookie;
function Add(const AName, AValue, APath: string): TCookie;
function Add(const AName, AValue, APath: string; Expiry: TDateTime): TCookie;

Four different overload functions that allow to add a new cookie to the TWebCookies collection.

property Items[Index: integer]: TCookie;

Property providing access to each cookie in the collection by an array indexer

```
function Find(const AName: string): TCookie;
```

Find a cookie instance by name in the collection

TWebClientConnector



The TWebClientConnector is a component to establish a connection between a TMS Web Core application running in the browser and a client application written in FMX or VCL running in a desktop or mobile environment. In combination with the TTMSFNCWebCoreClientBrowser (available in TMS FNC Core) the TMS Web Core application can be viewed in your favorite environment. TWebClientConnector is defined in the unit WEBLib.ClientConnector.

Setting up the TWebClientConnector

In your TMS Web Core application, drop an instance of TWebClientConnector on the form. There are no additional steps necessary to start receiving and sending messages at browser side.

To receive messages, you can implement the OnReceivedMessage event. The OnReceivedMessage returns JSON, below is a sample of parsing JSON in the OnReceivedMessage event:

```
procedure TForm1.DoReceivedMessage(Sender: TObject; AJSON:
TJSONObject);
var
  s: string;
begin
  s := TJSJSON.stringify(AJSON.JSONObject);
  WebMem1.Text := s;
end;
```

Sending messages with the TWebClientConnector

To send messages, you need to encapsulate your data in JSON, then send it to the client, which the TWebClientConnector is connected to.

```
procedure TForm1.SendButtonClick(Sender: TObject);
```

```
var
  o: TJSONObject;
  js: TJSON;
  s: string;
  I: Integer;
begin
  js := TJSON.Create;
  s := '{"Message From Browser":"' +
TTMSFNCUtils.EscapeString(WebMemol.Text) + '"}';
  o := js.parse(s);
  w.Send(o);
  o.Free;
end;
```

Ofcourse, sending and receiving will only work when a client, writing in VCL or FMX, is connected. Below are the steps necessary to have a working connection between browser and client.

Setting up the TTMSFNCWebCoreClientBrowser

Drop an instance of the TTMSFNCWebCoreClientBrowser on the form and enter the URL of your TMS Web Core application. When starting the application, the client will automatically try to establish a connection with the TMS Web Core application running the TWebClientConnector component instance. When the connection is established, the OnConnected event is triggered, allowing you to start sending and receiving messages. For receiving messages at client side, the OnReceivedMessage event (similar to the TMS Web Core application implementation for TWebClientConnector) can be used.

```
procedure TForm1.DoReceiveMessage(Sender: TObject; AJSON: TJSONValue);
var
  s: String;
begin
  if AJSON.TryGetValue<String>('Message From Browser', s) then
  begin
    ShowMessage(TTMSFNCUtils.UnescapeString(s));
  end;
end;
```

To send messages to the TMS Web Core application you can use the following code:

```
procedure TForm1.SendButtonClick(Sender: TObject);
var
```

```
    c: TJSONObject;  
begin  
    c := TJSONObject.Create;  
    c.AddPair('Message From Client', 'Hello World !');  
    TMSFNCWebCoreClientBrowser1.Send(c);  
    c.Free;  
end;
```

TWebAESEncryption

Below is a list of the most important properties, methods and events for the TWebAESEncryption class. The supported algorithms are: AES-CBC and AES-GCM.

Properties for TWebAESEncryption

Property	Description
AESType: TAESEncryptionType	The AES encryption algorithm type. If modified, it's not applied to the current key.
CryptoKey: TJSCryptoKey	The CryptoKey object.
ExtractableKey: Boolean	Determines if the key is extractable. If modified, it's not applied to the current key.
KeyLength: TAESEncryptionKeyLength	The key length. If modified, it's not applied to the current key.
Usages	Set of key usages. If modified, it's not applied to the current key.

Methods for TWebAESEncryption

Property	Description
Decrypt(AEncryptedData: TJSSArrayBuffer; AResultType: TCryptoDecryptResultType)	Method to decrypt an encoded data with the class's key. The result type can be string or binary, based on what kind of data was encoded.
Encrypt(APlainText: string)	Method to encrypt a plain text with the class's key.
Encrypt(ABinary: TJSSUint8Array)	Method to encrypt binary data with the class's key.
ExportKey(AFormat: TCryptoExportImportFormat)	Method to export the class's key. Supported formats are: raw (ArrayBuffer) and jwk (JSON string).
GenerateKey	Generates a new key based on the current property settings.
ImportKey(AJSON: string)	Method to import an AES key that is stored as a JSON string.
ImportKey(ABinary: TJSSUint8Array)	Method to import an AES key that is stored as binary data.
ImportKey(ARaw: TJSSArrayBuffer)	Method to import an AES key that is stored as an array buffer.

Property	Description
UnwrapKey(AlimportFormat: TCryptoExportImportFormat; AKey: TJSSArrayBuffer; AKeyAlgorithm: JSValue; AExtractable: Boolean; AKeyUsages: TCryptoKeyUsages)	Method to unwrap AKey with the class's key and algorithm. AKeyAlgorithm is the algorithm of AKey. AlimportFormat must be the same as what was used for wrapping.
WrapKey(AKey: TJSCryptoKey; AExportFormat: TCryptoExportImportFormat)	Method to wrap a key with the class's key and algorithm.

Events for TWebAESEncryption

Property	Description
OnDecryptedBinary	Event triggered when an encrypted data is decrypted and the format is binary.
OnDecryptedString	Event triggered when an encrypted is decrypted and the format is string.
OnEncrypted	Event triggered when a data is encrypted.
OnError	Event triggered when there's a Promise rejection.
OnKeyCreated	Event triggered when a key is created.
OnKeyExportedJSON	Event triggered when a key is exported as a JSON string.
OnKeyExportedRaw	Event triggered when a key is exported as an array buffer.
OnKeyImported	Event triggered when a key is imported.
OnKeyUnwrapped	Event triggered when a key is unwrapped.
OnKeyWrapped	Event triggered when a key is wrapped.

TWebRSAEncryption

Below is a list of the most important properties, methods and events for the TWebRSAEncryption class. The supported algorithm is: RSA-OAEP.

Properties for TWebRSAEncryption

Property	Description
----------	-------------

Property	Description
ExtractableKey: Boolean	Determines if the key is extractable. If modified, it's not applied to the current key.
Hash: TCryptoHash	The hash function to be used with the algorithm. If modified, it's not applied to the current key.
ModulusLength: TRSAModulusLength	The length in bits of the RSA modulus. If modified, it's not applied to the current key.
PrivateKey: TJSCryptoKey	The private CryptoKey object.
PublicKey: TJSCryptoKey	The public CryptoKey object.
Usages	Set of key usages. If modified, it's not applied to the current key.

Methods for TWebRSAEncryption

Property	Description
Decrypt(AEncryptedData: TJSArryBuffer; AResultType: TCryptoDecryptResultType)	Method to decrypt an encoded data with the class's private key. The result type can be string or binary, based on what kind of data was encoded.
Encrypt(APlainText: string)	Method to encrypt a plain text, with the class's public key.
Encrypt(ABinary: TJUInt8Array)	Method to encrypt binary data, with the class's public key.
ExportKey(AKeyType: TCryptoAsymKeyType; AFormat: TCryptoExportImportFormat)	Method to export the class's keys. AKeyType represents which key to export. The supported formats are PKCS#8 (PEM encoded string) for private keys, SPKI (PEM encoded string) for public keys, and jwk (JSON string) for private/public keys.
ImportKey(AKey: string; AKeyType: TCryptoAsymKeyType; AFormat: TCryptoExportImportFormat)	Method to import a string formatted key. AKeyType determines which key to import (private/public). AFormat should be PKCS#8/jwk in case of a private key and SPKI/jwk in case of a public key.
ImportKey(ABinary: TJUInt8Array; AKeyType: TCryptoAsymKeyType)	Method to import a key stored in binary format. Will automatically use PKCS#8 for a private key and SPKI for a public key.
GenerateKey	Generates a new key pair based on the current

Property	Description
	property settings.
UnwrapKey(AlimportFormat: TCryptoExportImportFormat; AKey: TJSArrayBuffer; AKeyAlgorithm: JSValue; AExtractable: Boolean; AKeyUsages: TCryptoKeyUsages)	Method to unwrap AKey with the class's private key and algorithm. AKeyAlgorithm is the algorithm of AKey. AlimportFormat must be the same as what was used for wrapping.
WrapKey(AKey: TJS CryptoKey; AExportFormat: TCryptoExportImportFormat)	Method to wrap AKey with the class's public key and algorithm.

Events for TWebRSAEncryption

Property	Description
OnDecryptedBinary	Event triggered when an encrypted data is decrypted and the format is binary.
OnDecryptedString	Event triggered when an encrypted is decrypted and the format is string.
OnEncrypted	Event triggered when a data is encrypted.
OnError	Event triggered when there's a Promise rejection.
OnKeyCreated	Event triggered when a key is created.
OnKeyExportedJSON	Event triggered when a key is exported as a JSON string.
OnKeyExportedPKCS8	Event triggered when a key is exported in PKCS#8 format as a PEM encoded string.
OnKeyExportedSPKI	Event triggered when a key is exported in SPKI format as a PEM encoded string.
OnKeyUnwrapped	Event triggered when a key is unwrapped.
OnKeyWrapped	Event triggered when a key is wrapped.
OnPrivateKeyImported	Event triggered when a private key is imported.
OnPublicKeyImported	Event triggered when a public key is imported.

TWebRSASignature

Below is a list of the most important properties, methods and events for the TWebRSASignature class. The supported algorithm is: RSASSA-PKCS1-v1_5.

Properties for TWebRSASignature

Property	Description
ExtractableKey: Boolean	Determines if the key is extractable. If modified, it's not applied to the current key.
Hash: TCryptoHash	The hash function to be used with the algorithm. If modified, it's not applied to the current key.
ModulusLength: TRSAModulusLength	The length in bits of the RSA modulus. If modified, it's not applied to the current key.
PrivateKey: TJSCryptoKey	The private CryptoKey object.
PublicKey: TJSCryptoKey	The public CryptoKey object.
Usages	Set of key usages. If modified, it's not applied to the current key.

Methods for TWebRSASignature

Property	Description
ExportKey(AKeyType: TCryptoAsymKeyType; AFormat: TCryptoExportImportFormat)	Method to export the class's keys. AKeyType represents which key to export. The supported formats are PKCS#8 (PEM encoded string) for private keys, SPKI (PEM encoded string) for public keys, and jwk (JSON string) for private/public keys.
GenerateKey	Generates a new key pair based on the current property settings.
ImportKey(AKey: string; AKeyType: TCryptoAsymKeyType; AFormat: TCryptoExportImportFormat)	Method to import a string formatted key. AKeyType determines which key to import (private/public). AFormat should be PKCS#8/jwk in case of a private key and SPKI/jwk in case of a public key.
ImportKey(ABinary: TJUInt8Array; AKeyType: TCryptoAsymKeyType)	Method to import a key stored in binary format. Will automatically use PKCS#8 for a private key and SPKI for a public key.
Sign(AText: string)	Sign AText with the class's public key and algorithm.
Sign(ABinary: TJUInt8Array)	Sign ABinary with the class's public key and algorithm.

Property	Description
Verify(ASignature: TJSArraryBuffer; AData: TJSArraryBuffer)	Verify AData with ASignature, using the class's private key and algorithm.

Events for TWebRSAJSignature

Property	Description
OnError	Event triggered when there's a Promise rejection.
OnKeyCreated	Event triggered when a key is created.
OnKeyExportedJSON	Event triggered when a key is exported as a JSON string.
OnKeyExportedPKCS8	Event triggered when a key is exported in PKCS#8 format as a PEM encoded string.
OnKeyExportedSPKI	Event triggered when a key is exported in SPKI format as a PEM encoded string.
OnPrivateKeyImported	Event triggered when a private key is imported.
OnPublicKeyImported	Event triggered when a public key is imported.
OnSigned	Event triggered when data is signed.
OnVerify	Event triggered when data is verified

TWebHMACSignature

Below is a list of the most important properties, methods and events for the TWebHMACSignature class.

Properties for TWebHMACSignature

Property	Description
CryptoKey: TJSCryptoKey	The CryptoKey object.
ExtractableKey: Boolean	Determines if the key is extractable. If modified, it's not applied to the current key.
Hash: TCryptoHash	The hash function to be used with the algorithm. If modified, it's not applied to the current key.
Usages	Set of key usages. If modified, it's not applied to the current key.

Methods for TWebHMACSignature

Property	Description
ExportKey(AFormat: TCryptoExportImportFormat)	Method to export the class's key. Supported formats are: raw (ArrayBuffer) and jwk (JSON string).
GenerateKey	Generates a new key based on the current property settings.
ImportKey(AJSON: string)	Method to import a HMAC key that is stored as a JSON string.
ImportKey(ABinary: TJUInt8Array)	Method to import a HMAC key that is stored as binary data.
ImportKey(ARaw: TJArrayBuffer)	Method to import a HMAC key that is stored as an array buffer.
Sign(AText: string)	Sign AText with the class's public key and algorithm.
Sign(ABinary: TJUInt8Array)	Sign ABinary with the class's public key and algorithm.
Verify(ASignature: TJArrayBuffer; AData: TJArrayBuffer)	Verify AData with ASignature, using the class's private key and algorithm.

Events for TWebHMACSignature

Property	Description
OnError	Event triggered when there's a Promise rejection.
OnKeyCreated	Event triggered when a key is created.
OnKeyExportedJSON	Event triggered when a key is exported as a JSON string.
OnKeyExportedRaw	Event triggered when a key is exported as an array buffer.
OnKeyImported	Event triggered when a key is imported.
OnSigned	Event triggered when data is signed.
OnVerify	Event triggered when data is verified

TWebPushNotifications

Below is a list of the most important properties methods and events for the TWebPushNotifications. Push notifications are tested and supported in: Chrome, Firefox, Firefox Developer Edition, Edge, Opera, and on Android: Chrome, Firefox, Opera, Samsung Browser.

Registration for push notifications

The RegisterServiceWorker procedure first registers the service worker, then automatically retrieves a subscription that is tied to that service worker. If the AutoRegisterSubscription property is set to True, then it automatically registered the subscriptions on the server via the given RegisterSubscriptionURL. The UserID is used as an identification, which means it should be unique to the user. At the same time a single UserID can be registered from different devices.

[view plain text](#)

```
1. procedure TForm1.WebButton1Click(Sender: TObject);
2. begin
3.     WebPushNotifications1.RegistrationUserID := 'UserID';
4.     WebPushNotifications1.RegisterServiceWorker;
5. end;
```

Multiple users on the same device

It's possible that there are multiple users who share the same device. They might be interested in different topics or the notifications are personalized and we want to avoid sending a notification to a user who is not entitled to see it (for example: email services). This can be resolved by introducing a login-logout mechanism. We can send the notifications as long as the user is logged in (= "Active"). For this purpose the data store has a UserActive boolean field which identifies if the user is active or not. By default this value is always set to True. If you'd like to modify this value, you can do so by using the Logout or Login methods.

[view plain text](#)

```
1. procedure TForm1.WebButton1Click(Sender: TObject);
2. begin
3.     //Request the server to set UserActive to False
4.     //for the given RegistrationUserID:
5.     WebPushNotifications1.RegistrationUserID := 'UserID';
6.     WebPushNotifications1.Logout;
7. end;
8.
9. procedure TForm1.WebButton2Click(Sender: TObject);
10. begin
11.     //Request the server to set UserActive to True
12.     //for the given RegistrationUserID:
13.     WebPushNotifications1.RegistrationUserID := 'UserID';
14.     WebPushNotifications1.Login;
15. end;
```

Properties for TWebPushNotifications

Property	Description
AutoGetSubscription: Boolean	Get the subscription automatically when the VAPID key is received.
AutoRegisterSubscription: Boolean	Automatically register subscription.
LoginURL: string	URL for setting the user's active state to True.
LogoutURL: string	URL for setting the user's active state to False.
Registration: TJSServiceWorkerRegistration	Provides access to the service worker registration object.
RegistrationUserID: string	A unique ID for the user (such as email).
RegistrationUserData: string	Used for setting topics. Use ',' as a separator between the topics.
RegisterSubscriptionURL: string	URL for registering the subscription.
ServiceWorkerURL: string	URL for the service worker.
Subscription: TJSPushSubscription	Provides access to the PushSubscription object.
UnregisterSubscriptionURL: string	URL for unsubscribing a subscription.
VapidPublicKey: string	VAPID public key. Can be fetched from the server using VapidPublicKeyURL.
VapidPublicKeyURL: string	URL to fetch the VAPID public key if it's not set yet.

Methods for TWebPushNotifications

Property	Description
CreateNewSubscription	Method to create a new subscription.
GetVapidPublicKey	Method to get the VAPID key from the server.
Login	Method to set the user's active state to True in the data store.
Logout	Method to set the user's active state to False in the data store.
RegisterServiceWorker	Method to register the service worker with the browser's push service. It creates a subscription if needed.
RegisterSubscription	Method to register a subscription on the server.
Unsubscribe(aAll: Boolean = True)	Method to unregister a subscription on the server. By

Property	Description
	default all the subscriptions will be unsubscribed that are connected to the UserID.

Events for TWebPushNotifications

Property	Description
OnGetRegistration	Event triggered when service worker registration is available.
OnGetSubscription	Event triggered when a subscription is available.s
OnGetVapidPublicKey	Event triggered when the VAPID public key is fetched from the server.
OnSubscriptionRegistered	Event triggered when a subscription is sucessfully registered.
OnUnsubscribed	Event triggered when the user has unsubscribed.

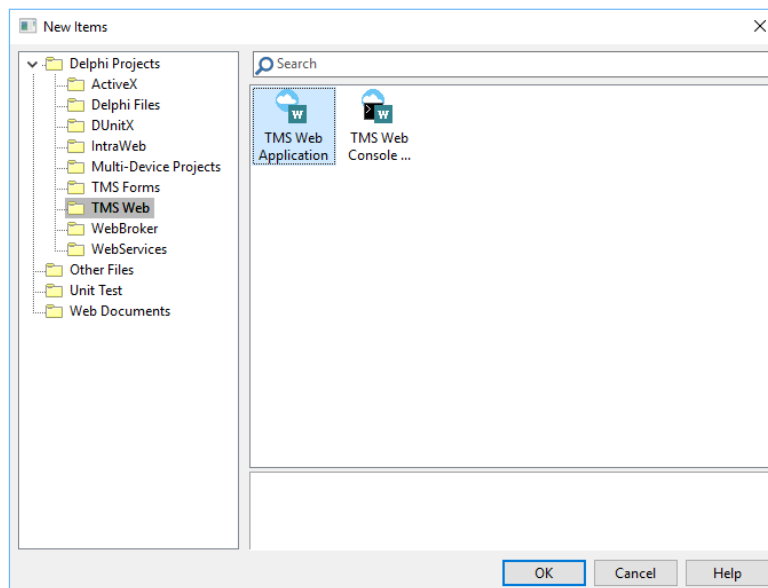
TMS WEB Core 3D

TMS WEB Core 3D component library can be used to create impressive 3D WebGL applications in Delphi. It consists of several components to display interactive 3D Charts and Models in a Web Application.

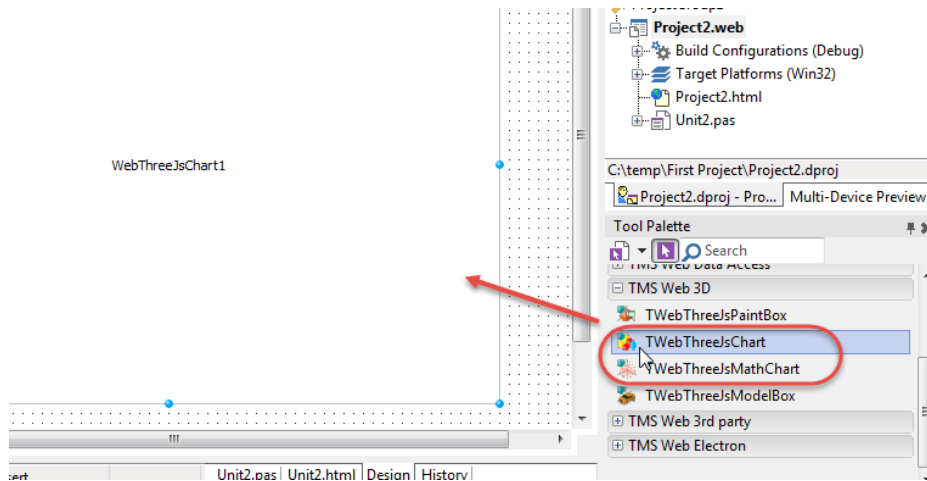
These components internally use WebGL through Three.js, an open source, cross-browser JavaScript library. The best thing is that one need not know WebGL or Three.js in order to make basic 3D applications with these components. At the same time, if the need arises, direct Three.js API calls can be made in Delphi code through a JS interface library provided for the purpose.

Your first 3D Chart application

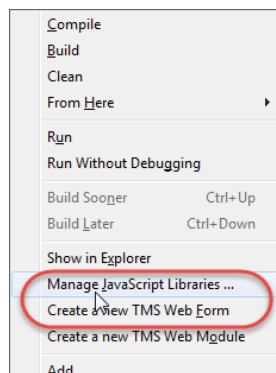
Create a standard TMS Web Application in the Delphi IDE by choosing File, New, Other, TMS Web Application.



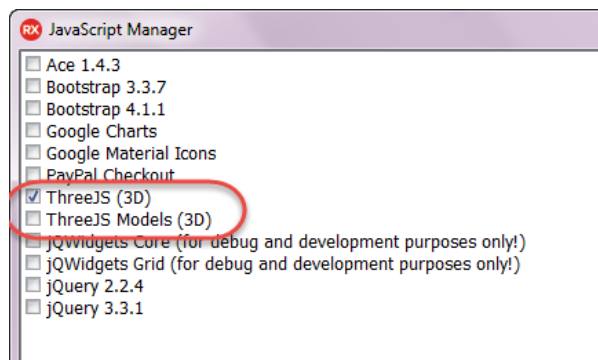
A new web form is created. Go to the Tool Palette and select the TWebThreeJsChart component from the “TMS Web 3D” section and drop it on the web form.



Now the only thing remaining is to include the proper Three.js library file. Right-click on the Project and select “Manage JavaScript Libraries.”



Choose the “Three JS (3D)” library and click OK.



Save and Run the project. You will see a Default Chart come up in the browser. Try to rotate the chart with the mouse and zoom with the mouse wheel. You get that interactive functionality out of the box, without writing any code!



You can now customize the data series for this chart in the code of the form as per your requirement. For some sample code, please see the demo project under the TMS folder Demo, 3D, Chart. The demo code is discussed in the next section.

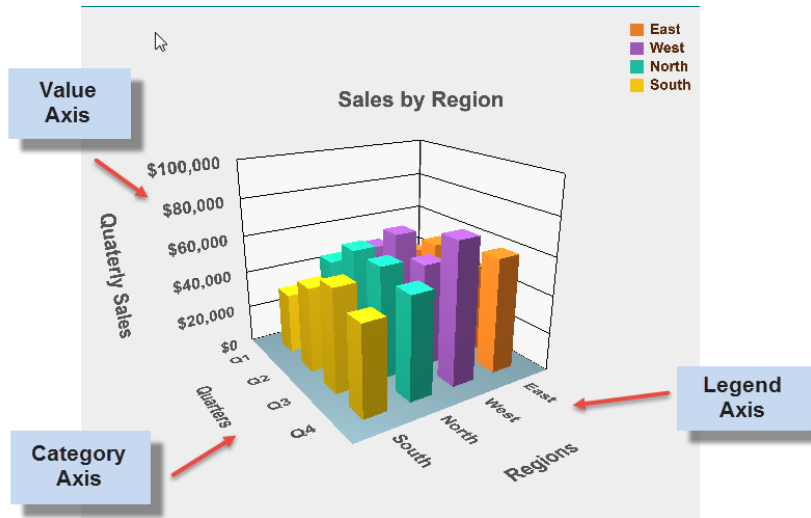
3D Business Chart Applications

As shown above, the **TWebThreeJsChart** component can be used to create 3D Business Chart applications that draw bar, line or area charts.

The 3D Bar Chart Demo

After creating a quick 3D chart application as shown in the previous section, the next step is to try and understand the customization code in the Chart Demo so that you can code your own custom data series for a similar 3D bar chart.

First of all, open the Chart Demo under the TMS folder Demo, 3D, Chart and run it. Move the mouse over the chart items and you will see them glow with a Text Popup showing the value. Try out various features given in the Demo before we discuss the code.



The Terminology for Axes

Before coding the data series, you will need to understand how each axis is named. The above picture shows the name of each Axis.

Creating the Data Series object

Please look at the Web Form code of the procedure LoadSampleSeries1.

```
var
  aSeries: TThreeJsChartSeries;
begin
  aSeries := TThreeJsChartSeries.Create(
    TJSArray.New('East', 'West',
      'North', 'South'),
    TJSArray.New('Q1', 'Q2', 'Q3',
      'Q4')
  );
```

The constructor of the Series object expects 2 parameters as the Axis Labels to be passed in two JS Arrays: 1. Legend Axis Labels 2. Category Axis Labels

Then the data is added in the form of each Legend Row as an array in the following code. The Demo uses hard coded data but you can have your own logic to obtain the data for each Legend row.

```
aSeries.addLegendRow('East',
  TJSArray.New(41834, 52835, 46563,
    60184));
aSeries.addLegendRow('West',
  TJSArray.New(48842, 62964, 54243,
    73796));
...
```

The rest of the Series set up code is easy to understand:

```
aSeries.valueAxisMarkMaximum := 100000;
aSeries.valueAxisMarkStep := 20000;
aSeries.valueFormatFloat := '$#,##0';
aSeries.valueAxisFormatFloat := ''; //use the above
aSeries.Title := 'Sales by Region';
aSeries.ValueAxisTitle := 'Quarterly Sales';
aSeries.LegendAxisTitle := 'Regions';
aSeries.CategoryAxisTitle := 'Quarters';
threeJsChart.Series := aSeries;
```

Notable points:

- The Format to show values on the chart items can be different from the format to show values on the Value Axis marks. But the above code uses the same format for both.
- The series object is finally assigned to the Series property of the Chart component. The above Load procedure is called from the WebFormCreate event and then the chart is displayed by the following code.

```
threeJsChart.clearChart;
threeJsChart.createChart;
```

The chart component is smart enough to decide on proper axes length and marks based on the data. But you can set the dimensions of items too, resulting in a bigger or smaller chart area.

Other features shown in the Demo

You can run and explore the Chart Demo to see many other features demonstrated:

- **Built in objects:** Many built in objects like the Camera, Spotlight, etc are automatically added by the component. You don't have to write any code.
- **Interactive Rotation and Zoom with the mouse and mouse wheel:** You get this functionality out of the box, without writing any code.
- **Choice of Chart Type:** Chart items can be shown as Bars, Cylinders, Cones, Lines or Areas.
- **Auto colors:** Colors are assigned automatically to Legend rows. You can specify custom colors too.
- **Dimension Properties for items:** Item Width, Space and Plot Width (for Line and Area charts) can be changed. The dimensions are in WebGL units. The component is smart enough to determine the length of bottom 2 axes automatically based on the item dimensions. But you can override the default length of Value Axis by a separate property.
- **Transparency of Chart Items:** can be set with additional Opacity property.
- **Optional Legend display:** can be specified to associate colors with Legend items.
- **Auto Marking of Value Axis:** The demo code does not use this feature by default. But you can switch on this checkbox to see its effect. This is a smart feature that determines the marks on the Value scale based on the data.

- **Other Niceties available:**
 - Auto rotate axis labels to always face the Camera
 - Show value popups on all items
 - Transparency and opacity control

Events

The following events are available:

- **Interaction with Items:** OnItemClick, OnItemExit, OnItemDbClick, OnItemMouseEnter, OnItemMouseLeave, OnItemMouseMove
- **Interaction with other areas of the chart:** OnClick, OnDbClick

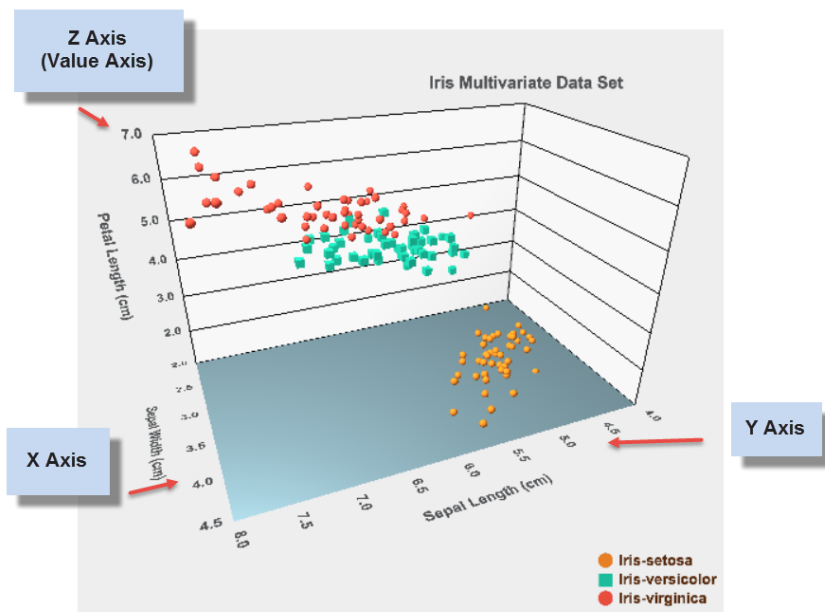
For example, the Demo uses the above events to show features like displaying value popups on items and glowing of items when the mouse is moved over the items or when the items are clicked.

3D Math Chart Applications

The component **TWebThreeJsMathChart** is available to create 3D Math Chart applications that draw Scatter or Surface charts.

The 3D Scatter Chart Demo

Open this Demo under the TMS folder Demo, 3D, Scatter and run it.



The Terminology for Axes

Before coding the data series, you will need to understand how each axis is named in a Math chart. This is different from the bar chart seen earlier that uses business terminology. The above picture shows the name of each Axis.

Creating the Data Series object

Please look at the Web Form code of the procedure LoadSampleSeries1. The Series class is **TThreeJsMathChartSeries**. The code that passes X, Y, Z data to the series is:

```
aSeries.addData(x, y, z, psSphere, name,  
                aPointSize, aPointColor)
```

In addition, the shape of the scatter point, its name, its size and color can be passed. The rest of the code to set the Series and to create the chart is similar to the earlier Chart Demo.

Other features shown in the Demo

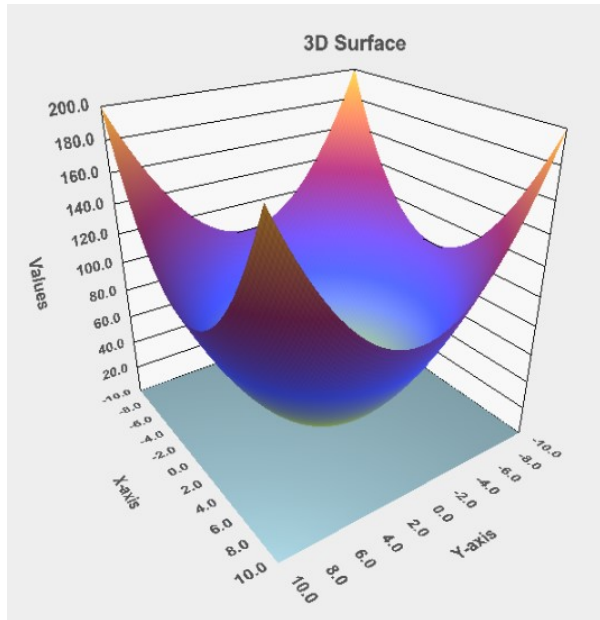
Other features demonstrated are similar to those described for the earlier Chart Demo except for the following differences:

- **Auto Marking** feature now determines the length and scale marks for all 3 axes based on the Series data. The Demo uses this feature. This feature saves considerable effort for a typical Math Chart application to pre determine and set the length and scale marks for each axis.
- **Improved Legend:** The Legend in the scatter chart shows the shape of the point in addition to the color.

The Events used in the Demo are also similar to those described for the earlier Chart Demo.

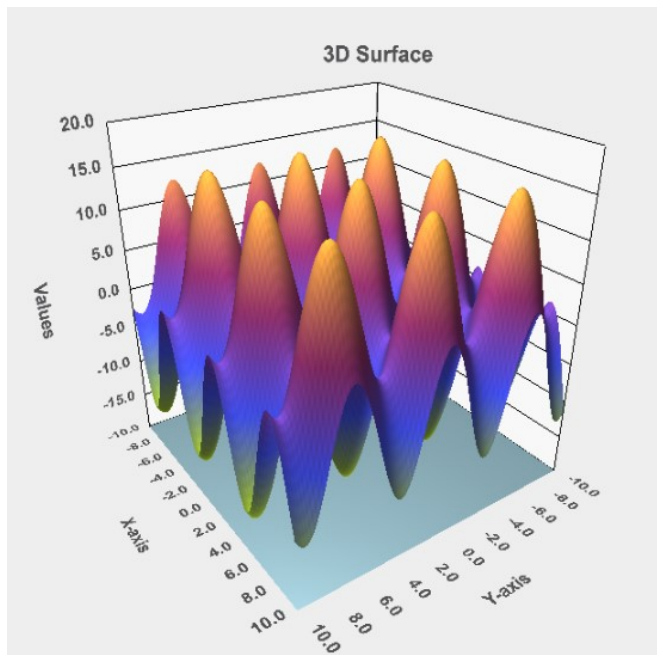
The 3D Surface Chart Demo

This demo demonstrates the features of **TWebThreeJsMathChart** component used to draw a surface chart based on an Equation. Hence, it is called a Parametric Surface Chart. Open this Demo under the TMS folder Demo, 3D, Surface and run it.



How it works

In the Demo, you can select a Surface Equation from a dropdown list to draw the surface chart accordingly. For example, here is another Surface Chart produced by the Demo:



Creating the Data Series object

Please look at the Web Form code of the procedure LoadEquation. The Series class is same as that for earlier Scatter Demo— **TThreeJsMathChartSeries**. But the procedure to add the data

for surface chart is different:

```
aSeries.addParametricSurface(xMin, xMax,
    yMin, yMax, resolution, @surfaceCallBack);
```

The ranges of values for X and Y are passed along with a Delphi Parametric callback function. The Chart component does the following:

- Generates X and Y values based on the parameters passed
- For each pair of values, calls back the Parametric function of the application to get the value of Z.

The callback function used in the Demo is:

```
function TForm2.surfaceCallBack(x, y: double): double;
begin
    case cbSeries.ItemIndex of
        1: Result := abs(x-y);
        2: Result := -x*x - y*y + 6;
        3: Result := sin(x)*x+cos(y)*y;
        4: Result := 2 * sqrt(x*x/3 + y*y/8);
        5: Result := sqrt(abs(1.5 * (x*x/3 -
            y*y/4) - 6));
        6: Result := 8 * (sin(x) + cos(y));
    else
        Result := x*x + y*y;
    end;
end;
```

The function uses the index of the drop down list to select an equation and return a Z value accordingly.

Other features shown in the Demo

You can run and explore the 3D Surface Chart Demo to see some more features provided for the Surface chart:

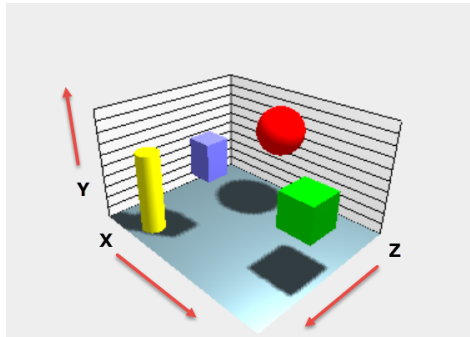
- **Show Wire Frame:** The cells of the wire frame depend on the Resolution value.
- **Show Wire Frame Texture:** This draws a wire frame texture directly on the colored surface of the chart. With the default high resolution, you may not be able to see this. Try a lower resolution value to see how this works.
- **Use Custom Colors and Texture.**

3D PaintBox Applications

The **TWebThreeJsPaintBox** component lets you create arbitrary 3D Scenes containing often used objects like Cubes, Spheres, Text and more. In fact, these primitives come from the base class and the already created Chart components are great examples of the kind of applications that are possible.

The 3D PaintBox Demo

You can open the PaintBox Demo under the TMS folder Demo, 3D, Paintbox and run it.



The Terminology for Axes

This component uses standard WebGL axes. These are shown in the picture above. You need to provide the positions of objects accordingly. All dimensions and positions are in WebGL units.

The code for adding objects

The objects are added to the scene in the OnCreate event of the Web Form with a code that looks like the following. The initial parameters for each object are dimensions, followed by the a TColor, followed by X, Y, Z position.

```
// Add a Bar with color $ff7777
anObject := threeJsPaintBox.AddBar(
    2, 3, 2, $ff7777, 8, 7, 10);
anObject.name := 'bar1';

// Add a cube
anObject := threeJsPaintBox.AddCube(3,
    $00FF00, 16, 5, 8);
anObject.name := 'cube2';

// Add a Sphere
anObject := threeJsPaintBox.AddSphere(2,
    $0000ff, 10, 8, 4);
anObject.name := 'sphere1';

// Add a Cylinder
anObject := threeJsPaintBox.AddCylinder(1,
```

```
1, 7, $00ffff, 4, 3.5, 13);
anObject.name := 'cylinder1';

// repaint the box to show new objects
threeJsPaintBox.Invalidate;
```

There are many more parameters for above functions with defaults, for example, to specify transparency. But the Demo code does not use them and default values are used for those. You can make quite impressive 3D applications by creating objects as shown above. The 3D Chart components discussed earlier are examples of such applications.

Direct Use of the Three.js API

Object creation methods like “AddCube” return a 3D Object of the type TThreeJsObject3D. Such types are Three.js objects made available to you in Delphi syntax via the specially coded JS Interface unit “Libthreejs.”

The end result is that you can directly use the methods and properties of these objects as documented in Three.js documentation. For example, to change the position of an object, you will change its “position” property directly, in Delphi code.

All the methods that expect a color as a parameter have been modified to use TColor of Delphi for the convenience of Delphi developers even though Three.js internally uses the Web color codes.

Sample code for Other features

Please run this Demo and inspect the source to see how you can perform these actions on objects in your own code.

- **Built in objects:** Many built in objects like the Camera, Spotlight, etc are automatically added by the component. You don't have to write any code.
- **Interactive Rotation and Zoom with the mouse and mouse wheel:** You get this functionality out of the box, without writing any code.
- **Rotation by code:** Rotate the whole scene around the origin by RotateLeft and RotateRight methods of the component. The demo uses Rotate trackbars to show this feature.
- **Change Center of Viewing:** To make another object's position as the center of viewing/rotation, use the method SetTargetViewVector and pass the position of another object. This is demonstrated by the button “Set Cube2 as Center of Viewing.”
- **Panning by code:** Similarly, you can pan the camera by Pan* methods of the component. The demo shows this feature by the 4 Pan buttons.
- **Zooming by code:** Use the ZoomIn method. This is shown by the Zoom trackbar.
- **Save orientation:** Suppose you want to save the exact orientation of the scene with respect to the camera and then restore it later. A SaveState method is provided for this purpose. Similarly, a ResetState restores the orientation to a saved state. The Demo shows this feature by “Save Orientation” and “Restore Orientation” buttons.
- **Debugging arrow:** Sometimes, you may want to know the exact position of certain invisible objects like the SpotLight and its target. A method ShowDebugArrow is

provided for this purpose. The demo shows this feature via the button “Show Arrow from Spotlight to Its Target.”

- **Moving an object:** If you have the handle of an object, you can change its position property directly. This is shown by the buttons “Move Spotlight Up/Down” where the object used is the Built In SpotLight object. You can use such code on any object that you have saved as a variable. The bottom trackbars are used in the Demo to move the object selected in the List “Operations on Object.”

Events

The following events are available:

- **Interaction with Objects:** OnItemClick, OnItemExit, OnItemDbClick, OnItemMouseEnter, OnItemMouseLeave, OnItemMouseMove
- **Interaction with other areas of the Scene:** OnClick, OnDbClick

3D Model Applications

The **TWebThreeJsModelBox** component lets you create or load arbitrary 3D Models from model files. In addition to the earlier described PaintBox methods, it contains methods to add Obj/Mtl or GLTF models from model files and can Export the scene to GLTF model files.

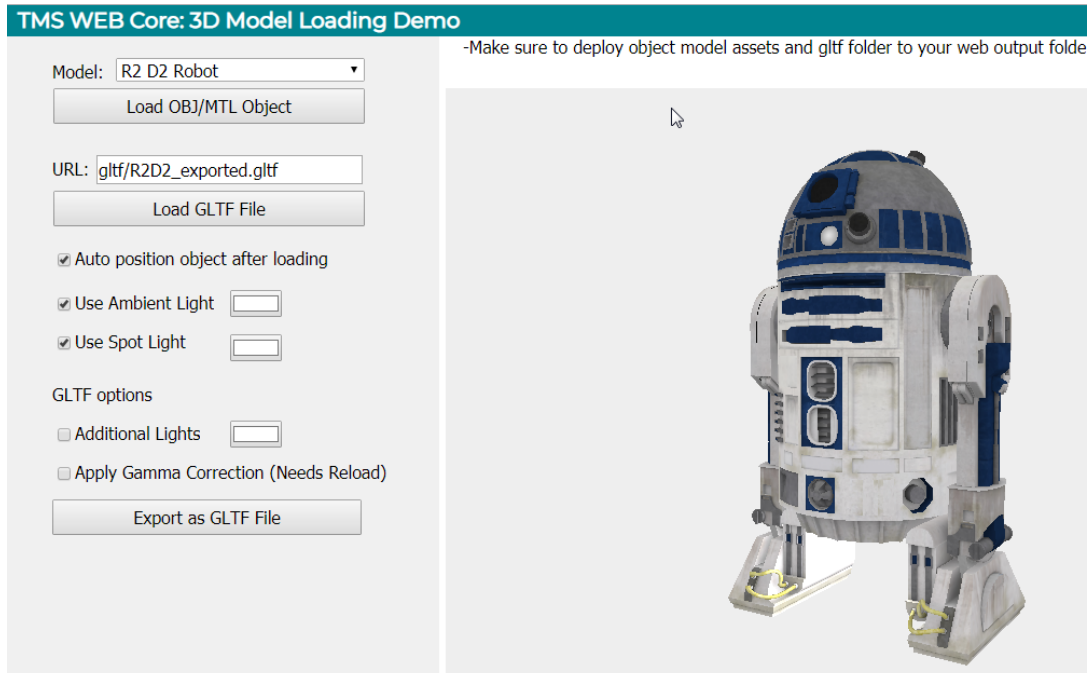
“ThreeJS Models (3d)” JS Library is required

If you use the TWebThreeJsModelBox component in an application, you should include the JS Library “ThreeJS Models (3d)” by using the same Project right-click menu “Manage JavaScript Libraries” that is described in an earlier section “Your first 3D Chart application.”

The 3D Model Demo

Since this Demo needs to load models from data files, an extra second step is required to copy those files as described below.

- Open the Model Demo under the TMS folder Demo, 3D, Model.
- Copy the 2 subfolders from the Data folder of the project to the web output folder.
- Now build and run the Demo.
- Select the Model “R2 D2 Robot” at the top and click on “Load Obj/Mtl Object” to get a result similar to the following picture.



The Code to Load OBJ/MTL Models

You will see a code like the following to add objects from OBJ/MTL files:

```
threeJsModelBox.AddObjectMtl('model-R2D2',
    'r2-d2.obj', 'assets/',
    'r2-d2.mtl', 'assets/',
    'assets/');
```

The first parameter is the name given to the model followed by the OBJ and MTL model file names and various folders accompanying them.

The code to Load GLTF Model

You will find this in the action code of Load GLTF Model button. The call is AddObjectGltf that is much simpler because only one GLTF file needs to be specified with an optional path for the accompanying folder, often not needed.

Getting the Object in OnObjectLoad event

Note that the loading of the Model is asynchronous and requires internal loading of many other files such as textures. So the above Add methods do not return an Object immediately. Instead, you have to use the event OnObjectLoad that hands over the object to you. There, you can take other actions on the object like rotating it if needed. For example, if you see the code

for this event, you will see a particular “Gothic Fence” object being rotated after loading because its default loaded view is horizontal, flat.

Additional features for Models

Auto Position Object after Loading by the method `BringObjectInFullView`

A third party 3d model object can be of any size. The Three.js code to properly position the camera so as to bring the object of any size in full view is complicated. Hence, the component implements a method **`BringObjectInFullView`** that does this job well.

This method is used in the Demo’s `OnObjectLoad` event to bring the object properly in view. You can see the difference made by this auto positioning method by unchecking the option when loading the R2 D2 model. The model is large and if you do not use the above method, you can only see the feet of the Robot after a load. You need to zoom out to see the full Robot which is big.

Additional Lights and Gamma Correction

To demonstrate loading of GLTF models, the Demo uses an already exported GLTF file from the Demo itself for the same R2 D2 Robot. Just click on the button “Load GLTF File” to load it. You will notice that the loaded model appears darker as compared to OBJ/MTL loaded result. This is so because GLTF models process the model as per their own algorithms needed to store everything in one file. Hence, GLTF models often need more light and something called a “Gamma Correction.” Hence, please switch on those options in the Demo, and then reload to see how it works better.

The above options use the component method `AddLights` and the property `UseGammaCorrection`.

Changing colors of lights

The Model Demo also shows sample code for changing color of various lights, including the built-in `SpotLight` and `AmbientLight`.

TWebMyCloudDbClientDataset Component

Introduction

The myCloudData.net service is an instantly available, secure and worry-free cloud data storage service.

The component TWebmyCloudDbClientDataset makes it easy for a Delphi TMS Web Application to use database tables on myCloudData.net service by a familiar syntax of using ClientDataSet. It also allows a seamless integration of the myCloudData.net data tables with data-aware components like TWebDBGrid. All the database operations can be done in the standard Delphi way through the TWebmyCloudDbClientDataset component.

All you need to do is specify the myCloudData properties and add the field definitions either in design more or in code in a standard Delphi syntax. Then connect a DataSource and Data components to it and make the dataset active.

Your first web application using TWebmyCloudDbClientDataset

Set up your myCloudData project in the myCloudData console

Follow these steps:

1. Navigate to <https://www.myclouddata.net/> and sign up for myCloudData if not already done
2. Go to My Account → Control Panel
3. Create a new Table and add the required Fields
4. Note the Table name. This will be used for the **TableName** property later. Go to My Account → API Key
5. Enter your myCloudData password and click “Get your App Key”
6. Note the App Key and App Callback URI values. These will be our properties **AppKey** and **AppCallbackURL** to be used later later. Note the App Secret value. This will be our property **AppSecret** to be used later. Note that the AppCallbackURL should be set to the URL of your web application. This can be different in debugging (typically something like <http://localhost:8000/Project1/Project1.html>) as from a deployed application. You might as such need to adapt the callback URL for deployment!

Create a TMS Web Application

Create a standard TMS Web Application in the Delphi IDE by choosing File, New, Other, TMS Web Application. A new web form is created.

Set up the TWebmyCloudDbClientDataset component

Go to the Tool Palette and select the TWebmyCloudDbClientDataset component from the “TMS Data Access” section and drop it on the web form.

Specify the Component Properties

Set up the properties either in code or in the Object Inspector by right-clicking on the “Fields Editor”:

- **AppKey**: from the “My Account → API Key” section of myCloudData.net
- **AppCallbackURL**: from the “My Account → API Key” section of myCloudData.net
- **TableName**: from the “My Account → Control Panel” section of myCloudData.net

Create the Fields or Properties of each object in the Object Store

The DataSet field definitions need to be set up either in Object Inspector by right-clicking on the “Fields Editor” or in the WebFormCreate event code.

Select the fields in the Object Inspector

Follow these steps:

1. Right-click the TWebmyCloudDBClientDataset and select “Fetch Fields”
2. Enter the Client ID (**AppKey**), Client Secret (**AppSecret**) and CallbackURL (A local URL is required here, for example: <http://127.0.0.1:8888>) values. Note that the TableName is retrieved automatically from the TableName property.
3. Click the “Fetch” button and follow the authentication instructions. If the process is successful, a dialog with the list of available fields is displayed.
4. Right-click the TWebmyCloudDBClientDataset and select “Fields Editor”
5. Select the required fields

Create the Fields in code

Here is an example of adding the field definitions in code in the OnCreate event. In the Object Inspector, double-click on OnCreate event of the Web Form. This creates an event handler procedure WebFormCreate. The following code in it sets up the field definitions. What fields you add are based on how you defined them for the Table in myCloudData.net. Note that _ID field must be defined as data type ftString.

```
1. myCloudClientDataSet.FieldDefs.Clear;
2. myCloudClientDataSet.FieldDefs.Add('_ID', ftString);
3. myCloudClientDataSet.FieldDefs.Add('note', ftString);
4. myCloudClientDataSet.FieldDefs.Add('date', ftDate);
5. myCloudClientDataSet.Active := True
```

Add Data Components that connect to the DataSet

Now select and drop a TWebDataSource, TWebDBGrid and TWebDBNavigator component on the Web Form.

Set up the DataSource and Data components

Set the DataSource's DataSet property to WebMyCloudDbClientDataset1. Then set the DataSource property of the grid and navigator to point to TWebDataSource1.

Set up the Columns of the DBGrid

Do that by clicking on the Columns property of the DBGrid.

Set up a New Record event

Since we will be adding New Records with the DB Navigator, we need to set up the default values of the record. For this, we set up an OnNewRecord event procedure for the myCloudDb Client Data Set in the Object Inspector and type the following code in it.

```
procedure TForm1..NewRecord(DataSet: TDataSet);
begin
    DataSet.FieldByName('note').AsString := 'New Note';
    DataSet.FieldByName('date').AsDateTime := Date; // set to today
end;
```

Run the Web Application

Now you can build and run the application. When you run it for the first time, the component automatically asks you to login by using your credentials for myCloudData.net. The DB Grid will appear empty as there are no records. Try adding new records with the Navigator and see how it works.

Todo List Demo

Please find this demo in the folder Demos. This Demo connects the component to a Tasks table to show you the Tasks with their status, description and dates.

Additional features in this Demo

Add, Update, Delete through separate data aware controls and buttons

The Demo allows you to perform add, update, delete operations through database field editor controls and buttons instead of through the Navigator.

Sorting on columns

We want to be able to sort on any column of the DB Grid by clicking on the header of the column. So we need to be able to read all the records in the order of that field. For this, we need to add a Sort Field Definition specifying the field to be sorted on. This is done in the event procedure GridTasksFixedCellClick.

```
1. myCloudClientDataSet.ClearSortFieldDefs;
2. myCloudClientDataSet.AddSortFieldDef(LIndex, gridTasks.Columns[ACol].SortIndicator = siAscending);
3.
4. myCloudClientDataSet.Refresh;
```

The first parameter to AddSortFieldDef call is the field name and the second parameter is a boolean flag that is true for ascending order and false for descending order. The Demo uses its own logic to pass this information and then Refreshes (reloads) the data in the desired order.

Updating, inserting and deleting data

This Demo also shows an example of connecting Data components like CheckBox or a Memo to the database so that those fields can be edited in the current record. After editing, a call to Update from the update button takes care of committing the changes to the cloud database. Similarly, the Demo has examples of Inserting a new record and Deleting the current record by respective calls.

Troubleshooting

Exceptions are displayed in a red alert message at the bottom of the web page. You can also look at the Browser Console for error messages.

If you start getting authentication errors when the application was working earlier, it's most probably a changed IP address. In any case, the first thing you can try is clear the Local Storage which is under Applications in Chrome Developer tools.

Reference Section

TWebMyCloudDbClientDataset

Below is a list of the most important properties and methods of TWebIndexedDbClientDataSet component.

Properties of TWebmyCloudDbClientDataSet

Property	Description
Active	Set this to True to activate the DataSet. Field definitions must be present along with other properties described below.
AppKey	Get from the "API Key" section of myCloudData.net.

Property	Description
AppCallbackURL	Get from the "API Key" section of myCloudData.net.
TableName	Specify a table name to connect to from the "Control Panel" section of myCloudData.net.
OnError	This is an event property that notifies the application of any errors from myCloudData.net. The event can be set up at design time in Object Inspector by double-clicking on it. If the Application does not subscribe to this event, an Exception is raised on such errors. If subscribed, the application can then decide what to do. For example, show error, raise exception or take some corrective action. Note that hard errors (Delphi Exceptions) are not passed in this event. Rather, they cause an Exception that appears in a red alert. But in any case, all errors are always logged to the browser console.

Methods of TWebmyCloudDbClientDataset

Only the methods specific to myCloudData are listed below. Other methods from the base DataSet classes are used in the standard way.

Refresh

procedure Refresh(Force: Boolean=False);

Refresh reloads all the objects from the database. If AddSortFieldDef has been used to set up sorting definitions, the objects are loaded in the order specified. In addition, the current record pointer is restored after the Reload which is convenient for the user interface of the web application. Refresh is internally postponed till all the pending updates started asynchronously are finished. The Force parameter ignores the pending updates and forces a reload.

AddSortFieldDef and ClearSortFieldDefs

Use AddSortFieldDef to add one or more sort definitions for loading the data. Before using a series of these calls, you must clear all sort definitions by calling ClearSortFieldDefs.

procedure AddSortFieldDef(aField: String; isAscending: Boolean));

Where

- aField - the field name for the sorting order
- isAscending - Set True for ascending order.

ClearTokens

After a successful authentication & authorization, the TWebmyCloudDbClientDataset will store the obtained access tokens in the local storage so that a next time, this does not need to be obtained again. If for some reason this needs to be removed, call

procedure ClearTokens;

TWebFirestoreClientDataset Component

Introduction

The component TWebFirestoreClientDataset makes it easy for a Delphi TMS Web Application to create and use database tables (called collections) on Google Cloud Firestore noSQL database by a familiar syntax of using ClientDataSet. It also allows a seamless integration of the Firestore data collections with data-aware components like TWebDBGrid. All the database operations can be done in the standard Delphi way through the TWebFirestoreClientDataset component.

All you need to do is specify the Firestore properties and add the field definitions either in design time or in code in a standard Delphi syntax. Then connect a DataSource and Data components to it and make the dataset active.

Your first web application using TWebFirestoreClientDataset

Set up your Firestore project in the Firebase console

Follow these steps:

1. Navigate to <https://console.firebase.google.com/> and sign up for Firebase if not already done
2. Create a new project in Firebase or select an existing project
3. In the left menu, select Database
4. Create a Firestore database. Choose the options “Start in test mode” and let the region be default
5. Don’t create a collection as our ClientDataSet component will create it if it doesn’t exist
6. Click on the tab “Rules” above and change the rules to allow only authenticated users to access the database:

```
rules_version = '2';
service cloud.firestore {
  match /databases/{database}/documents {
    match /{document=**} {
      allow read, write: if request.auth != null;
    }
  }
}
```

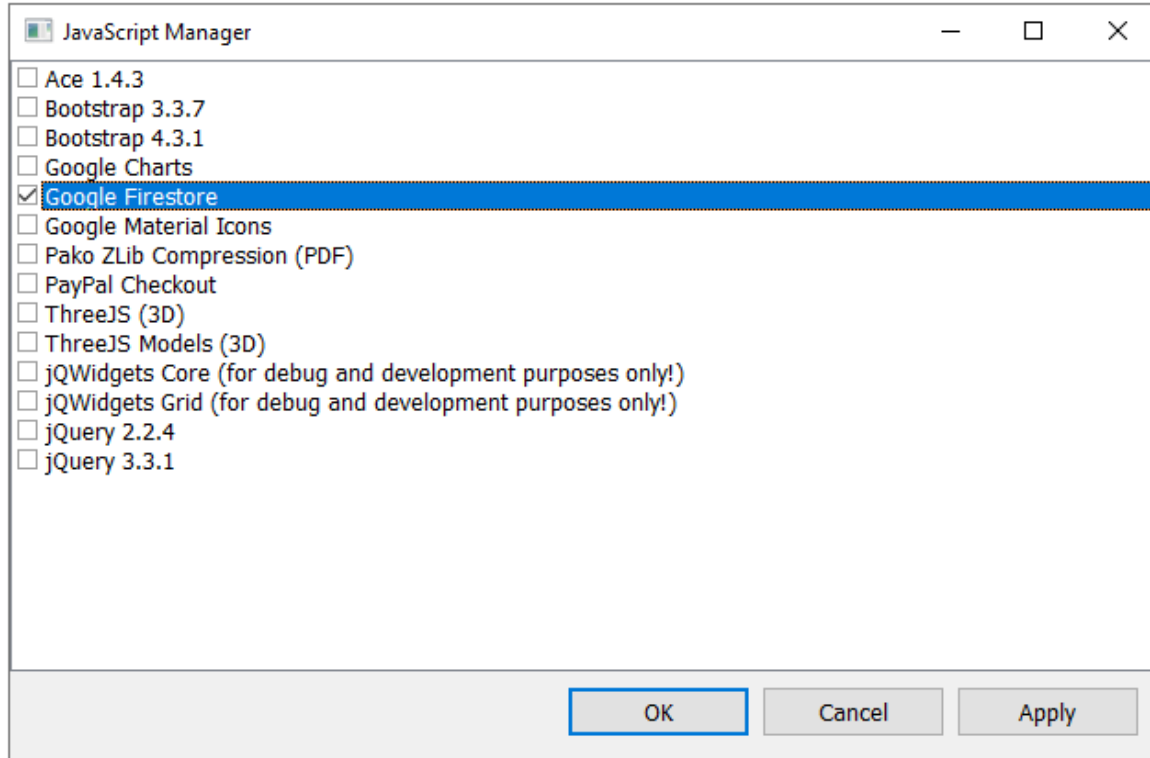
7. Click on Authentication in left menu and select Sign-in method as Google. Enable it. Note the authorized domain with firebaseapp.com. For example, test-15a3d.firebaseio.com. This will be our **AuthDomain** property to be used later.
If your TMS web application will run on localhost, make sure localhost is added to the list.
If your TMS web application will run on a remote webserver, make sure the domain name is added to the list.

8. Click on the Settings Gear Icon next to Project Overview on the left. Note the Project ID and Web API Key values. These will be our properties **ProjectId** and **ApiKey** to be used later.

Create a TMS web application

Create a standard TMS Web Application in the Delphi IDE by choosing File, New, Other, TMS Web Application. A new web form is created.

Enable the Firestore JavaScript libraries for your project. From the project context menu in the IDE, select “Manage JavaScript libraries” and select Google Firestore



Set up the TWebFirestoreClientDataset component

Go to the Tool Palette and select the TWebFirestoreClientDataset component from the “TMS Data Access” section and drop it on the web form.

Specify the Component Properties

Set up the properties either in code or in the Object Inspector as given below:

ApiKey: as obtained above in step 8 above.

AuthDomain: as obtained above in step 7 above.

ProjectId: as obtained above in step 8 above.

CollectionName: select a name of the collection that you want to use

KeyFieldName: specify the name of the key field

AutoGenerateKeys: set to True

SignInRequired: set to True as we set up this requirement in authentication rules above

Create the Fields or Properties of each object in the Object Store

The DataSet field definitions need to be set up either in code or in the Object Inspector by right-clicking on the “Fields Editor”.

Select the fields in the Object Inspector

Follow these steps:

1. Set up your Google App in the Google Developers Console
(<https://console.developers.google.com/>)
 - 1a. Go to “Credentials” → “Create Credentials” “Create OAuth client ID”
 - 1b. Select “Web Application”, enter the Authorized URL: <http://127.0.0.1:8888> and click “Create”
 - 1c. The Client ID and Client Secret values are displayed
 - 1d. Go to “Dashboard” and enable the required API(s)
2. Right-click the TWebFirestoreClientDataset and select “Fetch Fields”
3. Enter the Client ID, Client Secret and CallbackURL values from step 1. Note that the CollectionName and ProjectID are retrieved automatically from the CollectionName and ProjectID properties.
4. Click the “Fetch” button and follow the authentication instructions. If the process is successful, a dialog with the list of available fields is displayed.
5. Right-click the TWebFirestoreClientDataset and select “Fields Editor”
6. Select the required fields

Create the Fields in code

Here is an example of adding the field definitions in code in the OnCreate event. In the Object Inspector, double-click on OnCreate event of the Web Form. This creates an event handler procedure WebFormCreate. Type the following code in it that sets up the fields and then makes the DataSet active.

```
fireStoreClientDataSet.FieldDefs.Clear;
fireStoreClientDataSet.FieldDefs.Add(id', ftString);
fireStoreClientDataSet.FieldDefs.Add('note', ftString);
fireStoreClientDataSet.FieldDefs.Add(('date', ftDate);
fireStoreClientDataSet.Active := True
```

Add Data Components that connect to the DataSet

Now select and drop a TWebDataSource, TWebDBGrid and TWebDBNavigator component on the Web Form.

Set up the DataSource and Data components

Set the DataSource's DataSet property to WebFirestoreClientDataset1. Then set the DataSource property of the grid and navigator to point to TWebDataSource1.

Set up the Columns of the DBGrid

Do that by clicking on the Columns property of the DBGrid.

Set up a New Record event

Since we will be adding New Records with the DB Navigator, we need to set up the default values of the record. For this, we set up an OnNewRecord event procedure for the Client Data Set in the Object Inspector and type the following code in it.

```
procedure TForm1..NewRecord(DataSet: TDataSet);
begin
    DataSet.FieldByName('note').AsString := 'New Note';
    DataSet.FieldByName('date').AsDateTime := Today;
end;
```

Run the Web Application

Now you can build and run the application. When you run it in a browser that is not logged in to Google already, the component automatically asks you to login by using your Google credentials. The DB Grid will appear empty as there are no records. Try adding new records with the Navigator and see how it works.

Todo List Demo

Please find this demo in the folder Demos. This Demo connects the component to a Tasks table to show you the Tasks with their status, description and dates.

Additional features in this Demo

Add, Update, Delete through separate data aware controls and buttons

The Demo allows you to perform add, update, delete operations through database field editor controls and buttons instead of through the Navigator.

Sorting on columns

We want to be able to sort on any column of the DB Grid by clicking on the header of the column. So we need to be able to read all the records in the order of that field. For this, we need to add a Sort Field Definition specifying the field to be sorted on. This is done in the event procedure GridTasksFixedCellClick.

```
fireStoreClientDataSet.ClearSortFieldDefs;  
fireStoreClientDataSet.AddSortFieldDef(LIndex, gridTasks.Columns[ACol].SortIndicator = siAscending);  
fireStoreClientDataSet.Refresh;
```

The first parameter to AddSortFieldDef call is the field name and the second parameter is a boolean flag that is true for ascending order and false for descending order. The Demo uses its own logic to pass this information and then Refreshes (reloads) the data in the desired order.

Updating, inserting and deleting data

This Demo also shows an example of connecting Data components like CheckBox or a Memo to the database so that those fields can be edited in the current record. After editing, a call to Update from the update button takes care of committing the changes to the cloud database. Similarly, the Demo has examples of Inserting a new record and Deleting the current record by respective calls.

Troubleshooting

Normally, you will see any exceptions raised in a red alert message at the bottom of the web page. You can also look at the Browser Console for error messages.

For any debugging, if you need to browse or edit the actual collection on the Cloud, you can do that in Firestore console. Note that individual records or objects under a Collection are called Documents in Firestore terminology.

TWebFirestoreClientDataset reference

Below is a list of the most important properties and methods of TWebFirestoreClientDataset component.

Properties of TWebFirestoreClientDataset

Property	Description
Active	Set this to True to activate the DataSet. Field definitions must be present along with other properties described below.
ApiKey	Get from the “Project settings” section of Firebase console as described earlier
AuthDomain	Get from the Authentication section of Firebase console as described earlier
CollectionName	Specify a collection name to connect to in Firestore
KeyFieldName	Set the name of the primary key field
AutoGenerateKeys	Recommended to set to True to let Firestore generate keys for new records
ProjectId	Get from the “Project settings” section of Firebase console as described earlier
SignInRequired	Set to True if only authenticated users are allowed access as per the Rules set up for the database. In this case, the component automatically tries to login on the first access.
OnError	This is an event property that notifies the application of any errors from Firestore. The event can be set up at design time in Object Inspector by double-clicking on it. If the Application does not subscribe to this event, an Exception is raised on such errors. If subscribed, the application can then decide what to do. For example, show error, raise exception or take some corrective action. Note that hard errors (Delphi Exceptions) are not passed in this event. Rather, they cause an Exception that appears in a red alert. But in any case, all errors are always logged to the browser console.

Methods of TWebFirestoreClientDataset

Only the methods specific to Firestore are listed below. Other methods from the base DataSet classes are used in the standard way.

Refresh

procedure Refresh(Force: Boolean=False);

Refresh reloads all the objects from the database. If AddSortFieldDef has been used to set up sorting definitions, the objects are loaded in the order specified. In addition, the current record pointer is restored after the Reload which is convenient for the user interface of the web application. Refresh is internally postponed till all the pending updates started asynchronously are finished. The Force parameter ignores the pending updates and forces a reload.

AddSortFieldDef and ClearSortFieldDefs

Use AddSortFieldDef to add one or more sort definitions for loading the data. Before using a series of these calls, you must clear all sort definitions by calling ClearSortFieldDefs.

procedure AddSortFieldDef(aField: String; isAscending: Boolean));

Where * aField - the field name for the sorting order * isAscending - Set True for ascending order.

TWebRadServerClientDataset

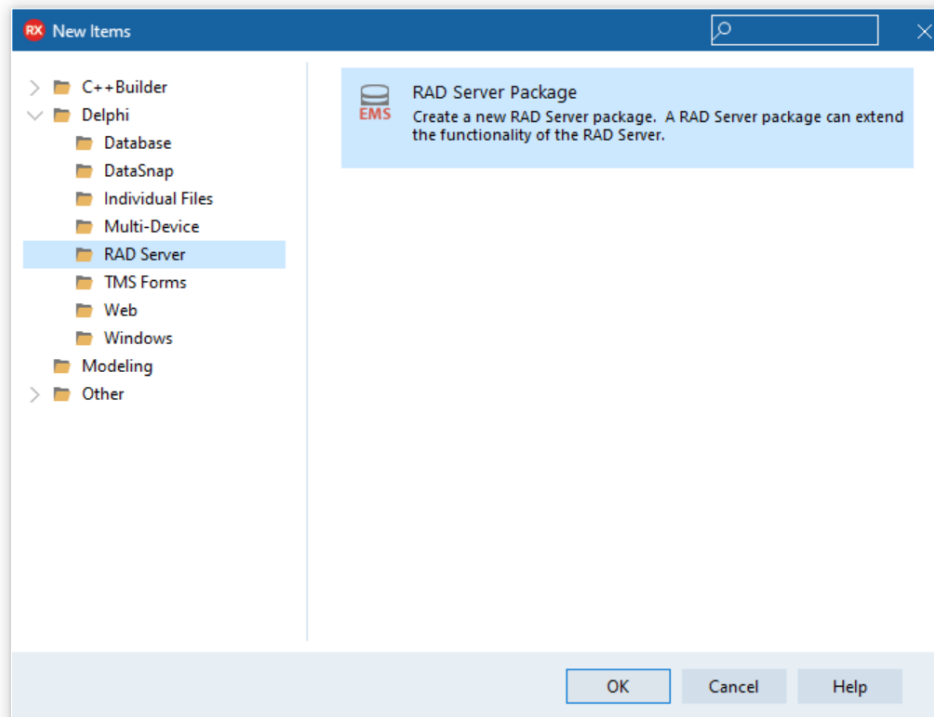


Introduction

Embarcadero Rad server (<https://www.embarcadero.com/products/rad-server>) is a technology for creating REST API services written in Delphi that can be hosted on Windows IIS or Linux Apache servers. These REST APIs can be accessed from TMS WEB Core web client applications. In its most basic form, the TWebHttpRequest component can be used to perform HTTP(s) GET,PUT,POST,DELETE requests to the APIs exposed by Embarcadero Rad Server. When creating a CRUD REST API functionality, the TWebRadServerClientDataset can internally fully handle the communication and offer access to the data via a TDataset based interface to the DB-aware UI controls in your web client applications. The TWebRadServerClientDataset is multi-tenant aware. This means that it works based on user-bound data, offers a login method and will perform operations on the data belonging to the logged-in user.

Configuring your Embarcadero Rad server back-end

Create a new Rad Server project from your Delphi IDE.



Create a new data module and set the `ResourceName` attribute to the name you want to use to access the dataset from the web client application. The `TWebRadServerClientDataset` will internally construct the URL to use the Rad Server REST API with.

To perform CRUD operation on a table, add methods `Get`, `Post`, `GetItem`, `PutItem`, `DeleteItem` to the datamodule:

```
[ResourceName('tasks')]
TTasksResource = class(TDataModule)
    conn: TFDConnection;
    query: TFDQuery;
published
    procedure Get(const AContext: TEndpointContext; const ARequest:
TEndpointRequest; const AResponse: TEndpointResponse);
    [ResourceSuffix('{item}')]
    procedure GetItem(const AContext: TEndpointContext; const ARequest:
TEndpointRequest; const AResponse: TEndpointResponse);
    procedure Post(const AContext: TEndpointContext; const ARequest:
TEndpointRequest; const AResponse: TEndpointResponse);
    [ResourceSuffix('{item}')]
    procedure PutItem(const AContext: TEndpointContext; const ARequest:
TEndpointRequest; const AResponse: TEndpointResponse);
    [ResourceSuffix('{item}')]
    procedure DeleteItem(const AContext: TEndpointContext; const
ARequest: TEndpointRequest; const AResponse: TEndpointResponse);
end;
```

In these methods return JSON objects for the `Get`/`GetItem` procedures from the data in the dataset used and get the posted data as JSON object and insert this as a new record in the dataset.

Note that as Rad Server is a multi-tenant architecture, the logged in user information can be retrieved from the `AContext` parameter of the methods. From here, `AContext.User.UserID` can be used to get the data belonging to a specific user or insert it with the correct user information.

The full source code for a sample Rad Server package that creates a REST API for CRUD operations on a tasks table can be found under `Demo\DBBackend\RadServer\Server`.

After creating and compiling the Rad Server package, follow these steps to start Rad Server with your package:

- 1) From the command line, execute:

```
EMSDevServer -l "RADServerTasks.bpl"
```

- 2) When you run this for the first time on a system
 - The EMSDevServer will not find any configuration and will ask you to Create it. Say YES.
 - Then Rad Server Setup Wizard starts. Do not change anything. Keep it at default. Note the location of DB File Directory because this is the location where it creates its EMS database for Rad Server and the INI file of parameters.
 - We are going to change the INI file there for our local Demo runs and tests.
C:\Users\Public\Documents\Embarcadero\EMS
 - Click Next and keep defaults for Sample Data where it will create sample users and user groups.
 - Click Next and note down the default user-name and password for IB Console, a utility.
 - Final screen asks for confirmation to create default files. Again leave them at default and click on Finish.
 - It shows some messages giving license warnings, etc. Once you are through, the compiled Rad Server starts running.
- 3) The Rad Server starts running. If you get an error that can not connect to EMS database then it means that Interbase service is not running. You will need to start it from Task Manager—Services
- 4) Once Rad Server is running, Click on Open Browser to do a quick test. It will show a version.
- 5) Change the URL in the browser to show tasks: <http://localhost:8080/tasks>
You will see JSON of the tasks present in the database. Once this works, you can start using the Client Demo that assumes that Rad Server is running on localhost:8080.
- 6) Stop the Server and close it.
- 7) EDIT the INI file **emsserver.INI** in the folder
C:\Users\Public\Documents\Embarcadero\EMS that we noted above.
Change the parameter CrossDomain's value to *. This will get rid of cross-domain error in Chrome that you would otherwise get.

CrossDomain=*

Now run the server again from the Batch file. Remember, whenever you change the INI file, you have to stop and restart the server.

Use Rad Server via TWebRadServerClientDataset

To start using the Rad Server REST API offering CRUD access to a table, drop a new TWebRadServerClientDataset instance on the TMS WEB Core web client application form.

- 1) Set the WebRadServerClientDataset.RadServerURL to the URL for the Rad Server. When performing local testing, this is default <http://localhost:8080>
- 2) Set the table name WebRadServerClientDataset.TableName, i.e. this is the ResourceName attribute set for the datamodule exposing the table.
- 3) Set the key field for the tasks table via WebRadServerClientDataset.KeyfieldName
- 4) Add the field types that will be used in the client dataset via WebRadServerClientDataset.FieldDefs

To login with a user account, use WebRadServerClientDataset.Login() passing the username and password. After a successful login, the dataset becomes active and any connected DB-aware control will show the data in the dataset. To signup a new user, just use the WebRadServerClientDataset.Login() method with last Boolean parameter set to true.

From this moment on, operations such as edit, insert, delete will be handled via the WebRadServerClientDataset on the Rad Server exposed table.

Reference

These are the properties, methods, events of the TWebRadServerClientDataset component

Properties

Property	Description
Active	Set this to True to activate the DataSet. Field definitions must be present along with other properties described below.
AppSecret	Sets the optional application secret key value
KeyfieldName	Sets the keyfield for the dataset
MasterSecret	Sets the optional master secret key value
RadServerURL	Sets the URL to perform the REST API HTTP(s) requests on
TableName	Sets the resource name that will be used in the Rad Server

Methods

Property	Description
Login(UserName,Password:string; IsSignup: boolean)	Performs a login on the Rad Server instance. When successful, the data is fetched in the dataset. When IsSignUp is true, a new account is created on the Rad Server instance

Events

The TWebRadServerClientDataset exposes the standard TDataSet events and is as such similar in functionality

TWebDreamFactoryClientDataset



Introduction

DreamFactory offers REST API creation without writing code. Via a web interface, the various characteristics of the REST API you want to create to let your application access data and other services on the back-end, can be configured. As such, you can create a REST API that can be consumed by a TMS WEB Core web client application. For handling CRUD operations on data that are exposed by a DreamFactory REST API, the TWebDreamFactoryClientDataset is available. The TWebDreamFactoryClientDataset is the bridge between the REST API and the DB-aware controls that are used in the web client application.

Configuring your DreamFactory back-end

Download the DreamFactory installer from <https://bitnami.com/stack/dreamfactory/installer> and install the software. After install, DreamFactory can by default be started via <http://127.0.0.1/dreamfactory/dist/index.html#/login>

To create a REST API service for a SQLite database used in the demo, follow these steps

Create the SQLite Service 'tasksdb'

- Select Services on the top menu
- Click Create on the left menu
- Click on "Service Type" dropdown to select Database--SQLite
 - Namespace: tasksdb
 - Label: Tasks DB Service
- Go to Config tab
 - Database: tasks
- Save

Create Schema Table Task

- Select Schema on the top menu

- Click on Service dropdown to select "Tasks DB Service". If not visible, click on Refresh button next to it or refresh the page.
- Click on upload JSON and upload the following JSON code. This will create the table.

```
{
  "resource": [
    {
      "name": "task",
      "label": "Task",
      "plural": "Tasks",
      "alias": null,
      "auto_increment": true,
      "is_primary_key": true,
      "field": [
        {
          "name": "id",
          "label": "Id",
          "type": "id"
        },
        {
          "name": "userid",
          "label": "User Id",
          "type": "user_id_on_update"
        },
        {
          "name": "status",
          "label": "Status",
          "type": "string",
          "db_type": "nvarchar(80)",
          "size": 80,
          "allow_null": false
        },
        {
          "name": "descr",
          "label": "Description",
          "type": "text",
          "allow_null": false
        },
        {
          "name": "due_date",
          "label": "Due Date",
          "type": "date",
```

```
        "allow_null": false
    }
]
}
]
```

Setup CORS

- Select Config on the top menu
- Click CORS on the left menu
- Click the + button
 - Path: *
 - Click on Methods dropdown to select "All"
 - Enabled: ON
- Save

Set up a Role "LoggedIn" to access the "tasksdb" service

- Select Roles on the top menu
- Click Create on the left menu
 - Name: LoggedIn
 - Active: ON
 - Go to Access tab
 - Click on + button to add a rule
 - Select Service as tasksdb
 - Select Component as *
 - Select Actions as All
 - Click on Show/Hide in the last column "Advanced filters". A Filter set up form appears. Click on + button to its right
 - Enter Field as userid
 - Leave Operator as =
 - Enter Value as {user.id}
- Save

Create App Tasks

- Select Apps on the top menu

- Click Create on the left menu
- Application Name: Tasks
- Click on "Default Role" dropdown to select our role created earlier, "Default"
- Active: ON
- Save
- Copy **the API key** from the Tasks app, it is required for the TMS WEB Core w.

Set up User service for Open Registration

- Select Services on the top menu
- Click on User service in the list
- Go to Config tab
 - Allow Open Registration: ON
 - Click on + button to add a Per App Open Reg Role
 - Select App as Tasks
 - Select Role as LoggedIn
- Click on "Open Reg Email Service" drop down and select the EMPTY value.
- Save

Using DreamFactory via TWebDreamFactoryClientDataset

Drop a TWebDreamFactoryClientDataset component on the form. First set the API key for the use of the DreamFactory REST API. This API key was obtained in the setup step 5 "Create App". Configure the URL of WebDreamFactoryClientDataset to the URL of the DreamFactory server. When testing on localhost, this is for example 'http://127.0.0.81'.

Setup the WebDreamFactoryClientDataset.DBServiceName to the name of the DreamFactory service you created, i.e. for this sample 'tasksdb' and set the WebDreamFactoryClientDataset.TableName to the name of the table you want to use for this dataset, i.e. for this sample 'tasks' and also set the unique key field name via WebDreamFactoryClientDataset.KeyfieldName.

Finally, setup the fields the dataset will use via the WebDreamFactoryClientDataset.FieldDefs.

Reference

These are the properties, methods, events of the TWebDreamFactoryClientDataset component

Properties

Property	Description
Active	Set this to True to activate the DataSet. Field definitions must be present along with other properties described below.
ApiKey	Sets the DreamFactory unique application secret obtained after creating a collection
DBServiceName	Sets the name of the service created in DreamFactory for accessing the database
DreamFactoryURL	Sets the URL for the DreamFactory server
KeyfieldName	Sets the name of the unique key field in the table
TableName	Sets the name of the table in DreamFactory to work with as a dataset for user management

Methods

Method	Description
Login(UserName,Password: string; IsSignup: boolean)	Performs a login on the DreamFactory service. When successful, the data is fetched in the dataset. When IsSignup is true, a new account is created on the Rad Server instance
Refresh	Reloads the data from DreamFactory table in the web client dataset
AddSortFieldDef(aField: string; isAscending: Boolean)	Sets the sort order of data returned from DreamFactory in the dataset
ClearSortFieldDefs	Clears any previously set sort order

Events

The TWebDreamFactoryClientDataset exposes the standard TDataSet events and is as such similar in functionality. It offers an additional event for handling a signup attempt with an already existing username

Event	Description
OnUserExists(Sender, UserName)	Event triggered when a signup is attempted using a username that already existed.

TWebFaunaDbClientDataSet



Introduction

FaunaDb offers a cloud hosted database and offers access to the data via a REST API. This REST API can be consumed by a TMS WEB Core web client application, allowing you to create web applications with data in the backend hosted by FaunaDb and as such no longer worry about this part of your web application. To make the use of the REST API from FaunaDb to perform CRUD operations on your data easier and codeless, the TWebFaunaDbClientDataset is available that works as a bridge between your FaunaDb hosted data and the DB-aware controls in your TMS WEB Core web client application.

Configuring the FaunaDb server back-end

To create a database with a table hosted on the FaunaDB cloud database, follow the steps below:

- 1) Sign up with Faunadb and Login to the dashboard
- 2) Click on "New Database" to create a new database with any name. Save.
- 3) In the left menu click on "Shell". From there execute the script to create the needed tables

Script step 1: create two tables

```
CreateCollection({ name: "Tasks" });
CreateCollection({ name: "users",
  permissions: { create: "public"}, //trying to allow sign up
});
```

Script step 2: create one user so login will be possible

```
Create(
  Collection("users"),
  {
    credentials: {password:"1234"},
    data: {
```

```
        email: "john@doe.com",
    },
}
);
```

Script step 3: create an index on the user table to sort users table

```
CreateIndex({
  name: "users_by_email",
  permissions: {read:"public"},
  source: Collection("users"),
  terms: [{field: ["data","email"]}],
  unique: true,
});
```

Script step 4: Assign admin privileges to logged in user to enable changing indexes for example

```
CreateFunction({
  name: "addPermission",
  role: "admin",
  body: Query(
    Lambda("x",
      Let (
        {curpriveleges:
          {"privileges":
            Union(Select("privileges",
Get(Role("loggedInUser"))),
          Var("x"))
        }
      },
      Update(Role("loggedInUser"),Var("curpriveleges"))
    )
  )
});
```

Script step 5: create the role that defines what the logged-in user can do

```
CreateRole({
  name: "loggedInUser",
  membership: {resource: Collection("users")},
```

- 4) At the end of the result, you should see a line with the Secret: "secret":
"xx"
- 5) Copy the value of the secret to the clipboard and use it in the web client application

To start using your created FaunaDB dataset, drop a TWebFaunaDBClientDataset instance on the TMS WEB Core web client application form.

- To login with a user account, use `WebFaunaDBClientDataset.Login()` passing the username and password. After a successful login, the dataset becomes active and any connected DB-aware control will show the data in the dataset.

393

WebFaunaDBClientDataset on the FaunaDB exposed table. To signup a new user, just use the WebFaunaDBClientDataset.Login() method with last Boolean parameter set to true.

Reference

These are the properties, methods, events of the TWebFaunaDbClientDataset component

Properties

Property	Description
Active	Set this to True to activate the DataSet. Field definitions must be present along with other properties described below.
ClientKey	Sets the FaunaDB unique application secret obtained after creating a collection
CollectionName	Sets the name of the collection in FaunaDB to work with as a dataset
KeyfieldName	Sets the keyfield for the dataset
MasterSecret	
UsersCollectionName	Sets the name of the user collection in FaunaDB to work with as a dataset for user management
UsersIndexName	Sets the name of the index for unique user identification

Methods

Property	Description
Login(Username,Password:string; IsSignup: boolean)	Performs a login on the FaunaDB cloud database. When successful, the data is fetched in the dataset. When IsSignup is true, a new account is created on the Rad Server instance
Refresh	Reloads the data from FaunaDB in the web client dataset
AddSortFieldDef(aField:string; isAscending: Boolean)	Sets the sort order of data returned from FaunaDB in the dataset
ClearSortFieldDefs	Clears any previously set sort order

Events

The TWebFaunaDBClientDataset exposes the standard TDataSet events and is as such similar in functionality. It offers an additional event for handling a signup attempt with an already existing username

Event	Description
OnUserExists(Sender, UserName)	Event triggered when a signup is attempted using a username that already existed.

TWebSQLRestClientDataset, TWebSQLRestConnection



Introduction

SQLDBRESTBridge is an open-source project <https://wiki.freepascal.org/SQLDBRestBridge> that offers a REST bridge for SQL databases. SQLDBRESTBridge allows to create a REST API for performing CRUD operations on a SQL database. Head to the wiki page for all details related to SQLDBRESTBridge.

Configuring the SQLDBRESTBridge server back-end

The demo comes with the source code to create the server instance (project restserver.lpr) that needs to be compiled with Lazarus as well as the executable restserver.exe.

The sample is based on a SQLite database todo.db. This database contains a user table and a tasks table. The tables are created with:

```
-- Fake autoincremental
create table t2(id integer primary key autoincrement);
-- These must match table names below !
insert into sqlite_sequence (name,seq) values ('Tasks',1);
insert into sqlite_sequence (name,seq) values ('Users',1);
drop table t2;

-- Primary key autoincrement, because this allows to assign a value
and will update sqlite_sequence
-- See https://www.sqlite.org/autoinc.html
create table Users (
    uID integer primary key autoincrement,
    uLogin varchar(50) not null,
    uPassword varchar(100) not null
);

create unique index udxUsers on Users(uLogin);

create table Tasks (
    tID integer primary key autoincrement,
    tUserFK integer not null,
```



```
tStatus varchar(15) not null,  
tDueDate date not null default CURRENT_DATE,  
tDescription varchar(4096)  
);  
  
create index idxTaskUser on Tasks(tUserFK);
```

After compiling the server, start it from the command line and it will be ready listening to requests on <http://localhost:8080/>

Using SQLite via TWebSQLRestClientDataset

Drop a TWebSQLRestConnection component on the form. This is the non-visual component through which the communication between the dataset and the REST server will happen. The TWebSQLRestConnection URI needs to be set to the URL of the server, in this case <http://localhost:8080/> The REST server can require a default login for which the credentials are set with TWebSQLRestConnection.User and TWebSQLRestConnection.Password.

Then drop two TWebSQLRestClientDataset instances on the form, one for the user table and one of the tasks table. The TWebSQLRestClientDataset metadata can be automatically initialized from the server or it can be programmatically done in the client. The example shows the Tasks table metadata being initialized from the server and the user table metadata programmatically initialized

```
// Retrieve metadata from server and setup indexes in the client for  
tasks table  
cdsTasks.UseServerMetaData := True;  
cdsTasks.Indexes.Add('ByDueDate', 'tDuedate', []);  
cdsTasks.Indexes.Add('ByStatus', 'Tstatus', []);  
cdsTasks.Indexes.Add('ByDescr', 'Tdescription', []);  
cdsTasks.Indexes.Add('ByDueDateDesc', 'tDuedate', [ixDescending]);  
cdsTasks.Indexes.Add('ByStatusDesc', 'Tstatus', [ixDescending]);  
cdsTasks.Indexes.Add('ByDescrDesc', 'Tdescription', [ixDescending]);  
cdsTasks.ActiveIndex := 'ByDueDate';  
cdsTasks.IDField:='tID';  
  
// programmatic field initialization for user table used to add new  
users
```

```
cdsnewuser.FieldDefs.Add('uID',ftLargeInt,0);  
cdsnewuser.FieldDefs.Add('uLogin',ftString,255);  
cdsnewuser.FieldDefs.Add('uPassword',ftString,255);
```

There is also a dataset `cdsValidLogin` used for the sole purpose of verifying the login.

The login is validated with:

```
CDSValidLogin.Close;  
CDSValidLogin.Params.ParamByName('uLogin').AsString :=  
edtLogin.text;  
CDSValidLogin.Params.ParamByName('uPassword').AsString :=  
edtPassword.text;  
CDSValidLogin.Load([], nil);
```

When opening this dataset, it either has or has not a record, indicating the user exists or does not exist. This is handled in the `cdsValidLogin.AfterOpen` event;

```
procedure TForm1.cdsValidLoginAfterOpen(DataSet: TDataSet);  
begin  
    FUID := -1;  
    if cdsValidLogin.Recordcount = 0 then  
        Showmessage('Invalid username/password')  
    else  
        begin  
            FUID := cdsValidLogin.FieldByName('uID').AsInteger;  
            EnableTasks;  
            LoadTasks;  
        end;  
end;
```

When the user is found, the user tasks dataset is loaded via `LoadTasks` and as the dataset is filled, the DB-aware UI controls can work on this dataset.

```
procedure TForm1.LoadTasks;  
begin  
    CDSTasks.Close;  
    CDSTasks.Params.ParamByName('uID').asInteger := fUID;  
    CDSTasks.Load([], nil);  
end;
```

The CDSTasks dataset then handles all further CRUD operations. In this sample, the client-side changes are not immediately updated in the server database. For this demo, it was chosen to do this in batch via calling CDSTasks.ApplyUpdates. The dataset will then internally handle applying all client-side dataset changes in one time to the server.

jQuery components

TMS WEB Core includes wrapper for the jqWidgets jQuery controls. This UI control can be obtained from: www.jqwidgets.com

To get started with the jqWidgets controls, it is important that the JavaScript and CSS libraries for these controls are added to the project. This is done by including the JavaScript libraries and CSS files to the main project HTML file. To get started, either open the main project HTML file from the Delphi IDE and add in the HTML file the script and CSS file references.

In the jqWidgets demo application, this is for example:

```
<!DOCTYPE html>
<html>
<head>
  <meta http-equiv="Content-type" content="text/html; charset=utf-8" />
  <link rel="icon" href="data:;base64,=">
  <title>TMS Web Project</title>
  <script type="text/javascript" src="TMSWeb_jqWidgets.js"></script>

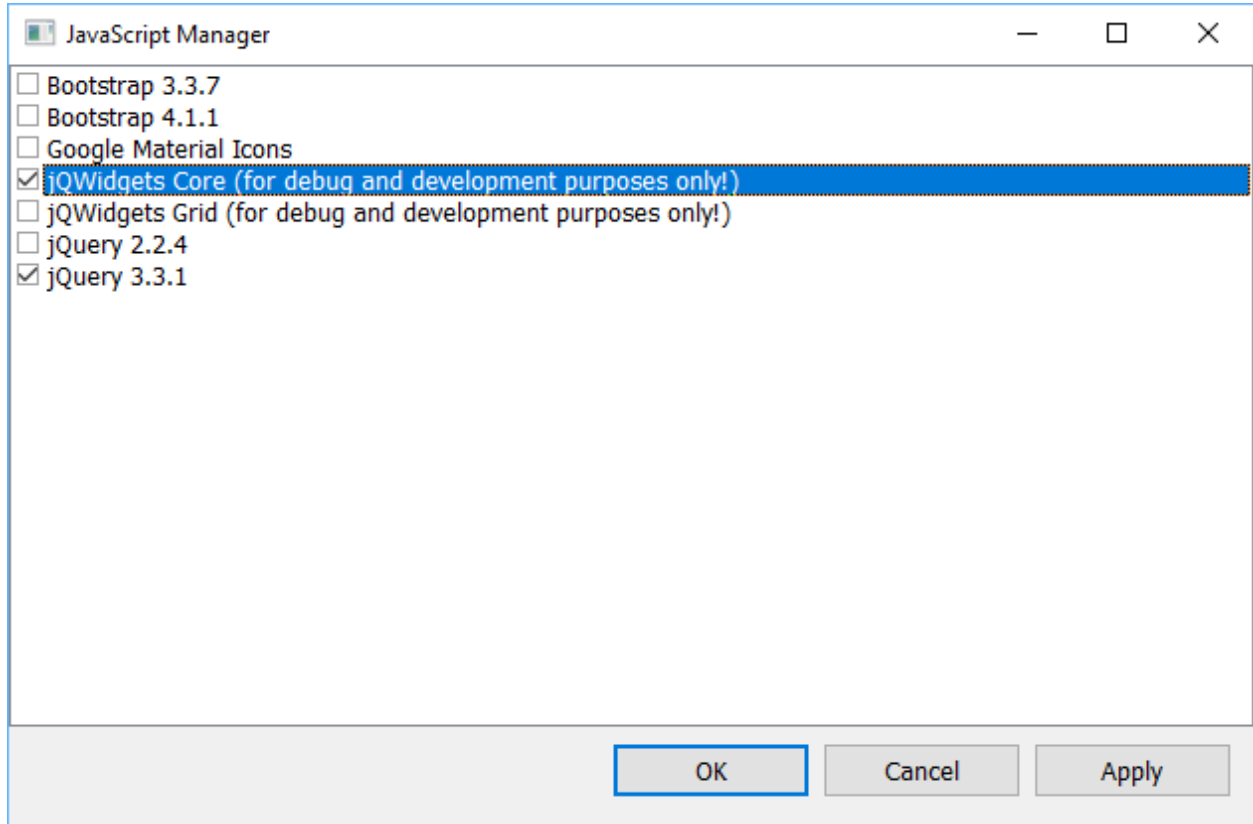
  <link rel="stylesheet" href="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/styles/jqx.base.css" type="text/css" />
  <link rel="stylesheet" href="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/styles/jqx.energyblue.css" type="text/css" />
  <link rel="stylesheet" href="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/styles/jqx.orange.css" type="text/css" />
  <link rel="stylesheet" href="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/styles/jqx.metrodark.css" type="text/css" />
  <link rel="stylesheet" href="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/styles/jqx.office.css" type="text/css" />

  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/scripts/jquery-1.11.1.min.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxcore.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxdatetimeinput.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxcalendar.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxmenu.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxmaskedinput.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxcolorpicker.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxscrollbar.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxbuttons.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxlistbox.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxcombobox.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxdropdownlist.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxnumberinput.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxrating.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxdraw.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxnob.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxprogressbar.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxsilder.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxbuttons.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxbuttongroup.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxradiobutton.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxrangeselector.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxdata.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxtagcloud.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxresponsivepanel.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/jqxtabs.js"></script>
  <script type="text/javascript" src="http://www.tmssoftware.biz/tmsweb/jqwidgets201804/jqwidgets/globalization/globalize.js"></script>

  <style>
  </style>
</head>
```

To make it easier for development and debugging, TMS WEB Core made a development version ready. To add jqWidgets UI control script references to your project, open the “Manage JavaScript libraries” menu item from the context menu in the Delphi IDE project manager and make sure to add first the jQuery 3.1.1 library reference followed by the jqWidgets development

library. There is a reference for the jqWidgets core UI controls and an additional separate reference for the jqWidgets grid:



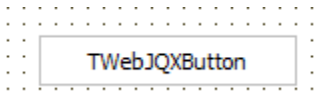
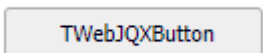
This adds the needed core jqWidgets library references to the project main HTML files. When you then add jqWidgets UI controls to the form, these controls will dynamically add their required additional jQuery files to the project HTML file.

Note that the jqWidgets library references added this way are for development purposes only! For a final release, it is required that you put the jqWidgets library files on your server and link to script files on your server!

TWebJQXButton

Description

Below is a list of the most important properties methods and events for TWebJQXButton.
Represents a button with optional image.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebJQXButton

Caption	Sets the caption text of the button
CaptionPosition	Sets the position of the caption text
CaptionImageRelation	Sets the position of the image relative to the caption text
ElementClassName	Optionally sets the CSS classname when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new control is created but

	the Delphi class is connected with the existing HTML element in the form HTML file
ImageUrl	Sets the URL of the image to be displayed in the button
ImagePosition	Sets the position of the image
ImageHeight	Sets the height of the image in pixels
ImageWidth	Sets the width of the image in pixels
RoundedBorders	Sets if the button is displayed with rounded borders
Template	Sets the template used to display the control. Options are Default, Primary, Success, Warning, Danger, Info
Theme	Sets the name of the theme that is used to display the control

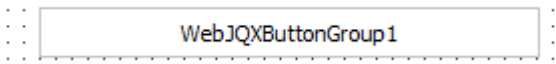
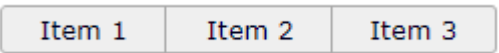
Events for TWebJQXButton

OnClick	Event triggered when the button is clicked
---------	--

TWebJQXButtonGroup

Description

Below is a list of the most important properties methods and events for TWebJQXButtonGroup. Represents a group of buttons. The buttons can optionally behave like a radio group or checkbox group.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebJQXButtonGroup

ButtonSelect[Button: Integer]	Select or unselect a button based on the index in the Items list
EnableHover	Enables the visual effect when a button is hovered
Items	A list of button caption texts
Mode	Sets how the button group behaves. Options are: Default, CheckBox, RadioButton
Template	Sets the template used to display the control. Options are Default, Primary, Success, Warning, Danger, Info

Theme	Sets the name of the theme that is used to display the control
-------	--

Events for TWebJQXButtonGroup

OnClick	Event triggered when a button is clicked
---------	--

TWebJQXCalendar

Description

Below is a list of the most important properties methods and events for TWebJQXCalendar. Represents a calendar that enables the user to select a date using a visual monthly calendar display.

WebJQXCalendar1

Designtime

April 2018

Mo	Tu	We	Th	Fr	Sa	Su
						1
2	3	4	5	6	7	8
9	10	11	12	13	14	15
16	17	18	19	20	21	22
23	24	25	26	27	28	29
30						

Runtime

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebJQXCalendar

Date	Sets the Calendar's Date. If multiselect is True this is the first day of range of dates
ElementClassName	Optionally sets the CSS classname when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new control is created but the Delphi class is connected with the existing HTML element in the form HTML file
EndDate	Sets the last day of a range of dates. Only if MultiSelect is True
FirstDayOfWeek	Sets which day to display in the first day column
MaxDate	Sets the maximum selectable date
MinDate	Sets the minimum selectable date
MultiSelect	If set to True a range of dates can be selected
OtherMonthDays	If set to True the days of days of the previous and next month are displayed
ShowToday	Sets if today's day is highlighted
Theme	Sets the name of the theme that is used to display the control
WeekNumbers	Sets if the week numbers are displayed

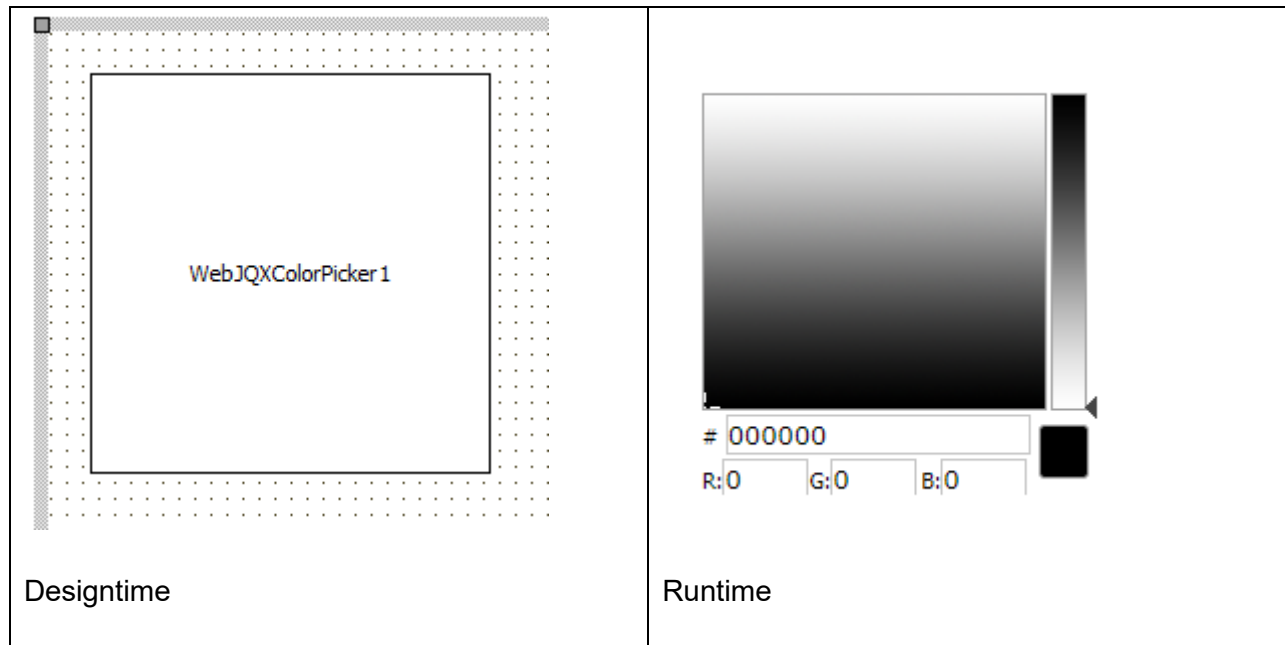
Events for TWebJQXCalendar

OnDateClick	Event triggered when a date is selected
OnNavigateClick	Event triggered when the calendar is navigated to a different month

TWebJQXColorPicker

Description

Below is a list of the most important properties methods and events for TWebJQXColorPicker. A control that allows the user to easily pick a color.



HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebJQXColorPicker

Color	Sets the selected color
ColorMode	Sets the color mode to hue or saturation

ElementClassName	Optionally sets the CSS classname when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new control is created but the Delphi class is connected with the existing HTML element in the form HTML file

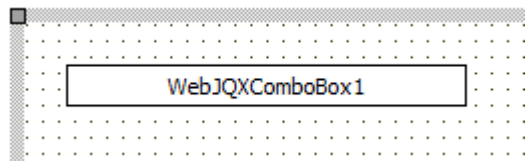

Events for TWebJQXColorPicker

OnChange	Event triggered when a color is selected
----------	--

TWebJQXComboBox

Description

Below is a list of the most important properties methods and events for TWebJQXComboBox. A combobox control that contains an input field with auto-complete functionality and a list of selectable items displayed in a drop-down.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebJQXComboBox

AutoComplete	If set to True only the items that match the searched text are displayed in the list
ElementClassName	Optionally sets the CSS classname when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new control is created but the Delphi class is connected with the existing HTML element in the form HTML

	file
ItemIndex	Sets the selected item index
Items	The collection of items
MultiSelect	Sets if multiple items can be selected
Theme	Sets the name of the theme that is used to display the control
TextHint	Sets the text displayed before an item is selected

Methods for TWebJQXComboBox

GetDisabled	Returns if the provided item index is disabled
SetDisabled	Sets the provided item index as disabled
GetSelected	Returns if the provided item index is selected
SetSelected	Sets the provided item index as selected

Events for TWebJQXComboBox

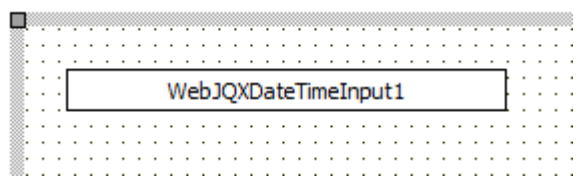

OnChange	Event triggered when an item is selected
----------	--

TWebJQXDateTimeInput

Description

Below is a list of the most important properties methods and events for TWebJQXDateTimeInput.

Represents a datetimeinput that enables the use to select a date or time using a popup calendar display or by keyboard input into the text field.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebJQXDateTimeInput

Date	Sets the Calendar's Date. If multiselect is True this is the first day of range of dates
ElementClassName	Optionally sets the CSS classname when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new control is created but the Delphi class is connected with the

	existing HTML element in the form HTML file
EndDate	Sets the last day of a range of dates. Only if MultiSelect is True
FirstDayOfWeek	Sets which day to display in the first day column
MaxDate	Sets the maximum selectable date
MinDate	Sets the minimum selectable date
MultiSelect	If set to True a range of dates can be selected
ShowToday	Sets if today's day is highlighted
Theme	Sets the name of the theme that is used to display the control
WeekNumbers	Sets if the week numbers are displayed

Events for TWebJQXDateTimeInput

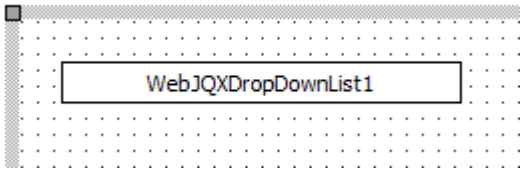

OnDateClick	Event triggered when a date is selected
-------------	---

TWebJQXDropDownList

Description

Below is a list of the most important properties methods and events for TWebJQXDropDownList.

Represents a control that contains a list of selectable items displayed in a drop-down.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebJQXDropDownList

ElementClassName	Optionally sets the CSS classname when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new control is created but the Delphi class is connected with the existing HTML element in the form HTML file
ItemIndex	Sets the selected item index

Items	The collection of items
MultiSelect	Sets if multiple items can be selected
Theme	Sets the name of the theme that is used to display the control
TextHint	Sets the text displayed before an item is selected

Methods for TWebJQXDropDownList

GetDisabled	Returns if the provided item index is disabled
SetDisabled	Sets the provided item index as disabled
GetSelected	Returns if the provided item index is selected
SetSelected	Sets the provided item index as selected

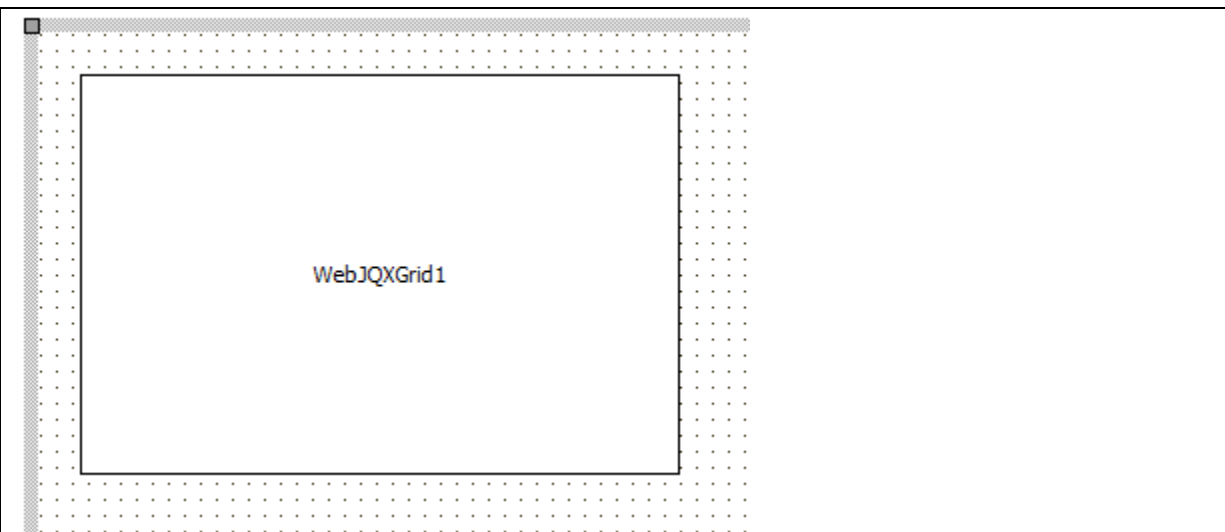
Events for TWebJQXDropDownList

OnChange	Event triggered when an item is selected
----------	--

TWebJQXGrid

Description

Below is a list of the most important properties methods and events for TWebJQXGrid. The Grid is a powerful control that displays tabular data. It offers rich support for interacting with data, including paging, grouping, sorting filtering and editing.



Designtime

Drag a column and drop it here to group by that column								
ID	Website	Image	Brand	Type	Date	Stock	Cc	KW
1	https://www.tmssoftware.com	WEB	Alfa Romeo	156 1.6TS	3/6/2018	<input checked="" type="checkbox"/>	1598	
2	https://www.tmssoftware.com	WEB	Alfa Romeo	156 1.8TS	3/7/2018	<input type="checkbox"/>	1774	1
3	https://www.tmssoftware.com	WEB	Alfa Romeo	156 2.0TS	3/8/2018	<input checked="" type="checkbox"/>	1970	1
4	https://www.tmssoftware.com	WEB	Alfa Romeo	156 2.5	3/9/2018	<input type="checkbox"/>	2492	1
5	https://www.tmssoftware.com	WEB	Alfa Romeo	166 2.0TS	3/10/2018	<input checked="" type="checkbox"/>	1970	1
6	https://www.tmssoftware.com	WEB	Alfa Romeo	166 2.0V6	3/11/2018	<input checked="" type="checkbox"/>	1996	1
7	https://www.tmssoftware.com	WEB	Alfa Romeo	166 2.5V6	3/12/2018	<input checked="" type="checkbox"/>	2492	1
8	https://www.tmssoftware.com	WEB	Alfa Romeo	166 3.0V6	3/13/2018	<input checked="" type="checkbox"/>	2959	1

Runtime

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebJQXGrid

Columns	
Alignment	Sets the text alignment in the column
ColumnType	Sets the type of the column. Default, Image (image URL displayed as an image), Email (email address displayed as hyperlink) or Link (URL displayed as hyperlink)
DataField	Sets the field name of the dataset field to bind the column to
DataType	Sets the datatype of the column. Available types are: Date, Double, Integer, String
Editor	Sets the editor for the column. Available editors are: CheckBox, DateTimeInput, DropDownList, Edit, None, NumberInput
Format	Sets the column formatting
Freeze	Sets if the column is fixed
Title	Sets the title of the column
Width	Sets the width in pixels of the column
Cells[Col, Row: Integer]	Gets or sets the value of a grid cell based on the column and row index
Data	

dataArray	If DataType is set to Array, assign a TJSArray with the data to load in the grid
DataType	Sets the type of data to load in the grid. Available types are: Array, CSV, JSON, None
Delimiter	Sets the delimiter character if DataType is set to CSV
Id	Sets the column name to be used as ID column if DataType is set to JSON
JSON	If DataType is set to JSON, assign the JSON data to load in the grid
Url	Assign the location of a CSV or JSON file to load the data in the grid if DataType is set to CSV or JSON respectively
RowSelect[Row: Integer]	Select a grid row based on the row index
FocusedCell	Gets or sets the currently focused cell
ElementClassName	Optionally sets the CSS classname when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new control is created but the Delphi class is connected with the existing HTML element in the form HTML file
Options	
Bands	
Enabled	Sets if row banding is enabled in the grid
RowCount	Sets the number of rows between banding rows
Editing	Sets if editing is enabled in the grid
Filtering	Sets if filtering is enabled in the grid
Grouping	Sets if grouping is enabled in the grid
Hovering	Sets if hovering is enabled in the grid

Paging	
Enabled	Sets if paging is enabled in the grid
PageSize	Sets the number of rows per page
SelectionMode	
Sets the selection mode. Options are single row, single cell, multiple rows, multiple cells	
Sorting	
ColumnIndex	Sets the column index the grid should be sorted by
Direction	Sets the sortdirection. Options are Ascending, Descending or Unsorted
Enabled	Sets if sorting is enabled
RowCount	
Sets the number of displayed rows	
RowHeight	
Sets the height of a grid row	
Theme	
Sets the name of the theme that is used to display the control	

Methods for TWebJQXGrid

SelectCell	Selects a single cell based on the provided row and column index
SelectRow	Selects a single row based on the provided row index

Events for TWebJQXGrid

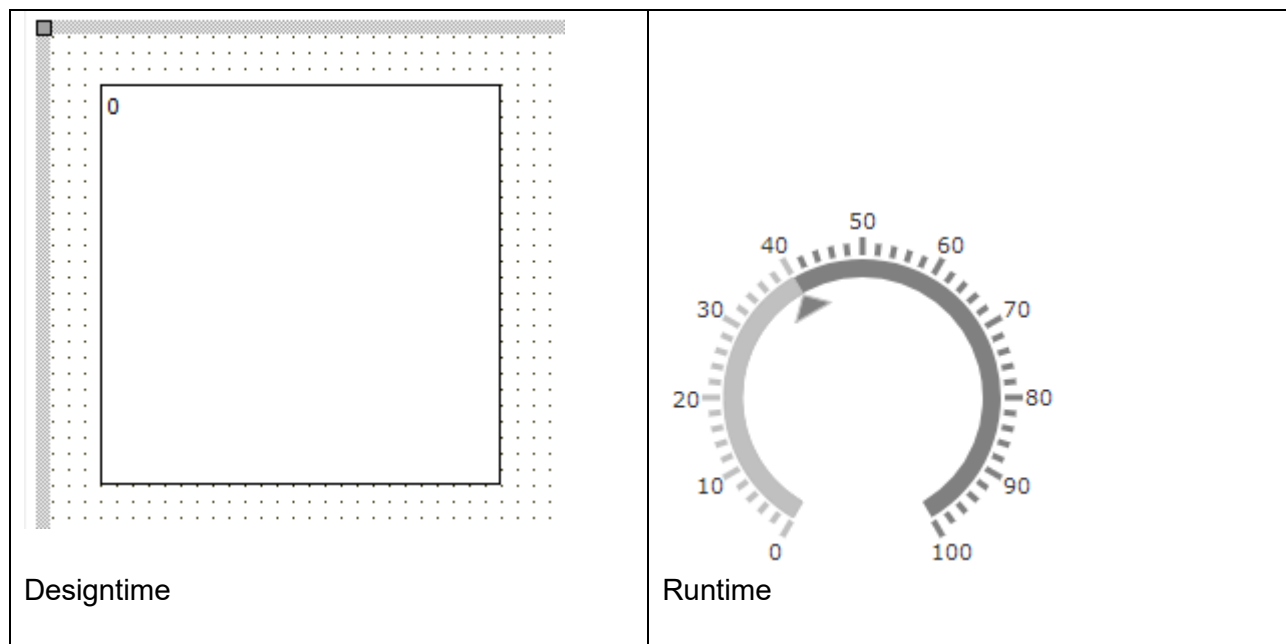
OnCellClick	Event triggered when a cell is clicked
OnCellEditClick	Event triggered when a cell is edited
OnCellEditDone	Event triggered after a cell is edited

OnCellEditStart	Event triggered when a cell is edited
OnCellEditValidate	Event triggered after a cell is edited
OnCellSelect	Event triggered when a cell is selected (via keyboard arrow keys)
OnFilter	Event triggered when the grid is filtered
OnGetCellData	Event triggered when a cell is rendered
OnPageChange	Event triggered when changing to a different page
OnRowClick	Event triggered when a row is clicked
OnRowSelect	Event triggered when a row is selected (via keyboard arrow keys)
OnSort	Event triggered when the grid is sorted

TWebJQXKnob

Description

Below is a list of the most important properties methods and events for TWebJQXKnob.
Represents a control with a round shape which displays a draggable indicator within a range of values.



HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebJQXKnob

Appearance

BorderColor	Sets the border color of the control
BorderWidth	Sets the border width of the control
Color	Sets the background color of the control
Labels	
Offset	Sets the labels position offset in percentage
Step	Sets the step between labels
Visible	Sets if the labels are displayed
Marks	
BorderColorProgress	Sets the border color of the marks in the progress part
BorderColorRemaining	Sets the border color of the marks in the remaining part
ColorProgress	Sets the color of the marks in the progress part
ColorRemaining	Sets the color of the marks in the remaining part
MajorInterval	Sets the interval between major marks
MajorSize	Sets the size of the major marks
MarkType	Sets the type of marks displayed. Options are Line or Circle
MinorInterval	Sets the interval between minor marks
Offset	Sets marks position offset in percentage
Size	Sets the size of the marks
Width	Sets the width of the marks
Pointer	
BorderColor	Sets the border color of the pointer
Color	Sets the color of the pointer
Offset	Sets the pointer position offset in percentage
PointerType	Sets the type of pointer displayed. Options are Arrow, Circle, Line
Size	Sets the size of the pointer

Visible	Sets if the pointer is displayed
Width	Sets the width of the pointer
ProgressBar	
BackgroundColor	Sets the background color of the progressbar
BorderColor	Sets the border color of the progressbar
Color	Sets the color of the progressbar
Offset	Sets the progressbar offset position in percentage
Size	Sets the size of the progressbar
EndAngle	Sets the ending angle of the progressbar for the maximum value
Maximum	Sets the maximum value
Minimum	Sets the minimum value
StartAngle	Sets the starting angle of the progressbar for the minimum value
Step	Sets the step between values in the range
Value	Sets the default value

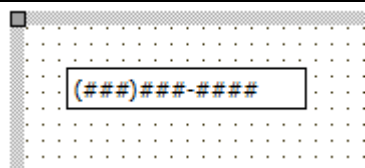
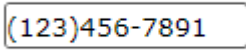
Events for TWebJQXKnob

OnChange	Event triggered when the value is changed
----------	---

TWebJQXMaskedInput

Description

Below is a list of the most important properties methods and events for TWebJQXMaskedInput. Represents an input control which uses a mask to distinguish between proper and improper user input.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebJQXMaskedInput

ElementClassName	Optionally sets the CSS classname when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new control is created but the Delphi class is connected with the existing HTML element in the form HTML file
Mask	Sets the mask configuration.

	# For an integer character from 0 to 9 A For an alpha numeric character from 0 to 9 and from A to Z L For an alpha character from A to Z
Text	Set the default text that is displayed
Theme	Sets the name of the theme that is used to display the control

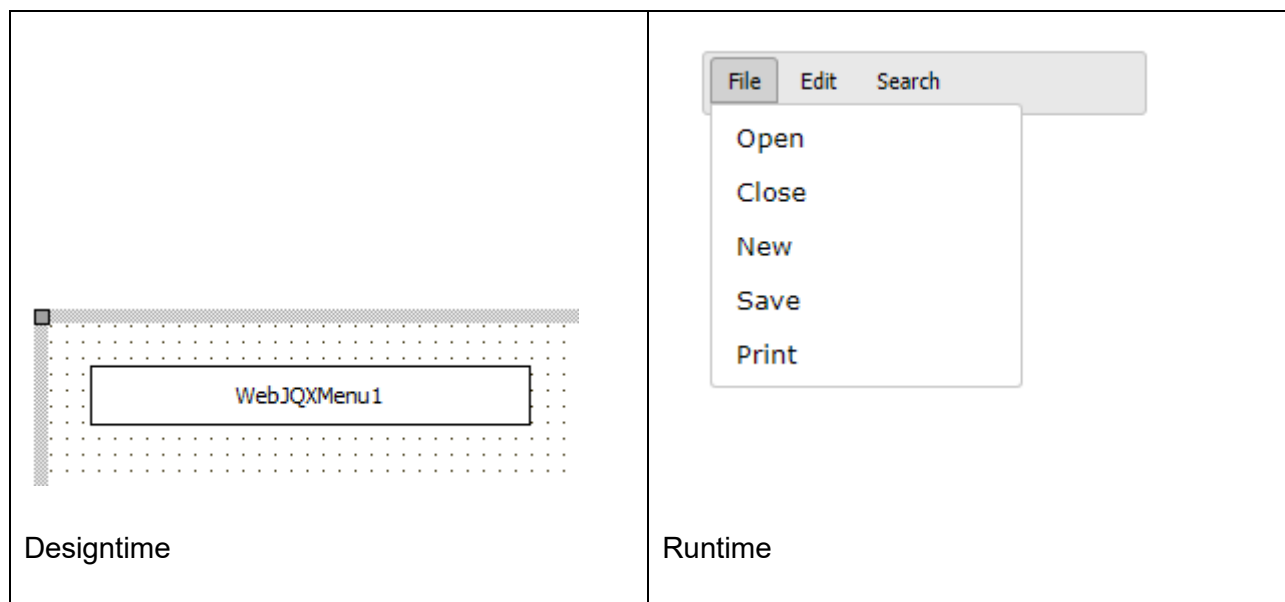
Events for TWebJQXMaskedInput

OnChange	Event triggered when the value is changed
----------	---

TWebJQXMenu

Description

Below is a list of the most important properties methods and events for TWebJQXMenu. Represents a menu control with support for sub-menus, it can be displayed vertical, horizontal or as a popup.



HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebJQXMenu

ElementClassName	Optionally sets the CSS classname when
------------------	--

	styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new control is created but the Delphi class is connected with the existing HTML element in the form HTML file
Menu	Set the TWebMainMenu control associated with the TWebJQXMenu
Mode	Set the display mode. Horizontal, Popup or Vertical
Theme	Sets the name of the theme that is used to display the control

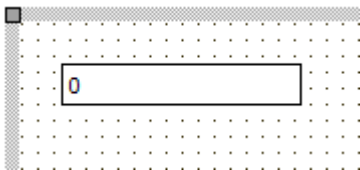

Events for TWebJQXMenu

OnItemClick	Event triggered when a menu item is clicked
-------------	---

TWebJQXNumberInput

Description

Below is a list of the most important properties methods and events for TWebJQXNumberInput. Represents a control that allows the user to input currency, percentages and any type of numeric data.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebJQXNumberInput

DecimalDigits	Sets the number of available decimal digits
Digits	Sets the number of available digits
ElementClassName	Optionally sets the CSS classname when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new control is created but the Delphi class is connected with the existing HTML element in the form HTML

	file
InputMode	Sets the input mode to Advanced (input with numeric mask) or Simple (restricted user input)
MaxValue	Sets the maximum allowed input value
MinValue	Sets the minimum allowed input value
ShowSpinButtons	Sets if the spin buttons are displayed
SpinButtonsStep	Sets the increase/decrease step
Symbol	Sets the character to use as currency or percentage symbol
SymbolPosition	Sets the position of the symbol. Left or Right
Theme	Sets the name of the theme that is used to display the control

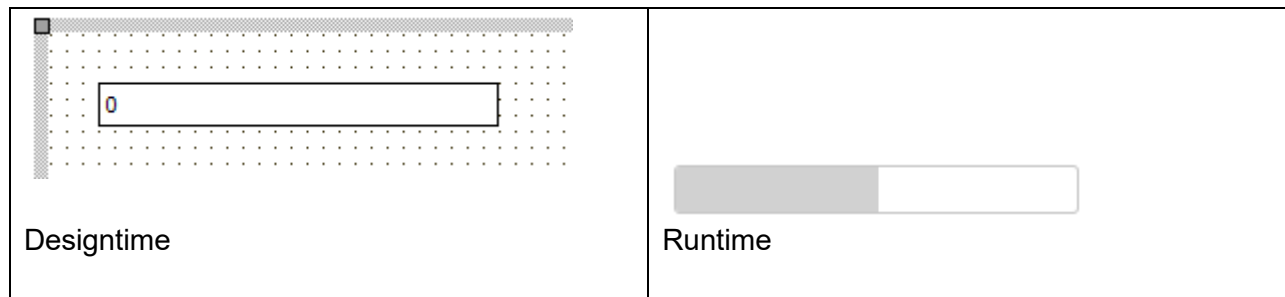
Events for TWebJQXNumberInput

OnChange	Event triggered when the input value is changed
----------	---

TWebJQXProgressBar

Description

Below is a list of the most important properties methods and events for TWebJQXProgressBar. Represents a control that visually indicates the progress of an operation.



HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebJQXProgressBar

AnimationDuration	Sets the duration of the animation to fill the progressbar to the value. Set to 0 to disable animation
ColorRanges	
Color	Sets the color of the progressbar up to the value set in the Stop property
Stop	Sets the end position for this color
ElementClassName	Optionally sets the CSS classname when


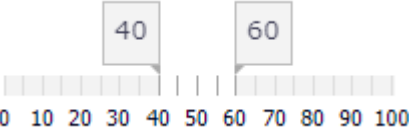
	styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new control is created but the Delphi class is connected with the existing HTML element in the form HTML file
Maximum	Sets the maximum value
Minimum	Sets the minimum value
Orientation	Sets the progressbar orientation to Horizontal or Vertical
ShowValue	Sets if the value is displayed in the progressbar
Theme	Sets the name of the theme that is used to display the control
Value	Sets the value of the progress position

TWebJQXRangeSelector

Description

Below is a list of the most important properties methods and events for TWebJQXRangeSelector.

Represents a control that can be used to select a numeric range.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebJQXRangeSelector

Maximum	Sets the maximum value of the range
MaximumValue	Sets the end value of the selected range
Minimum	Sets the minimum value of the range
MinimumValue	Sets the start value of the selected range
MajorTicksInterval	Sets the interval between major ticks
MinorTicksInterval	Sets the interval between minor ticks
MoveOnClick	Sets if the range is moved left or right when the range selector is clicked
Resizable	Sets the if the initial range can be resized by dragging the thumbs

ShowMajorTicks	Sets if the major ticks are displayed
ShowMinorTicks	Sets if the minor ticks are displayed
ShowMarkers	Sets if the markers (thumbs) are displayed
Theme	Sets the name of the theme that is used to display the control

Events for TWebJQXRangeSelector

OnChange	Event triggered when the range is changed
----------	---

TWebJQXRating

Description

Below is a list of the most important properties methods and events for TWebJQXRating. Represents a control that allows to select a rating.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebJQXRating

ElementClassName	Optionally sets the CSS classname when styling via CSS is used
ElementID	Optionally sets the HTML element ID for a HTML element in the form HTML file the label needs to be connected with. When connected, no new control is created but the Delphi class is connected with the existing HTML element in the form HTML file
ItemCount	Sets the number or rating items displayed
Value	Sets the default value

Events for TWebJQXRating

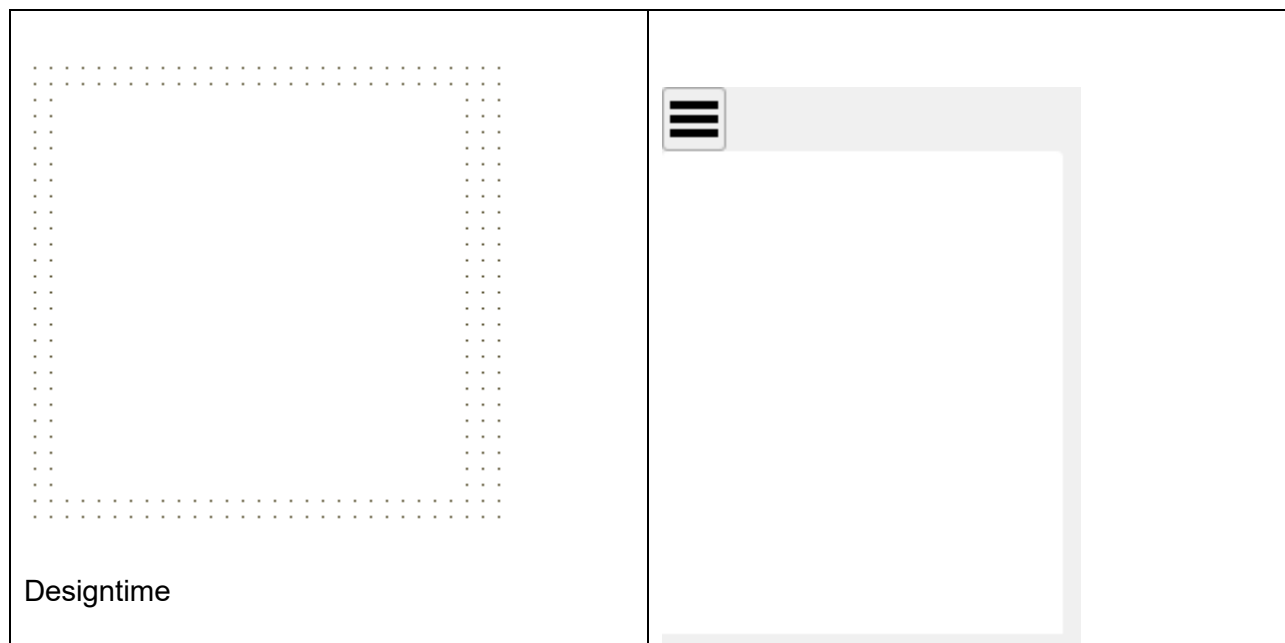
OnChange	Event triggered when the value is changed
----------	---

TWebJQXResponsivePanel

Description

Below is a list of the most important properties methods and events for TWebJQXResponsivePanel.

Represents a panel control with a responsive behaviour. The responsive panel collapses when the browser window (or parent element) width becomes less than a set value and the panel is then accessible by clicking a button.



	Runtime
--	---------

Properties for TWebJQXResponsivePanel

AnimationType	Sets the type of animation used when the panel is opened or closed. Options are Fade, Slide or None
AutoClose	Sets if the panel is automatically closed when a mouse click occurs outside the panel (only while the ToggleButton is visible)
CollapseBreakPoint	If the width of the browser window (or parent element) in pixels is lower than this value the ToggleButton is displayed, otherwise the ToggleButton is hidden
Theme	Sets the name of the theme that is used to display the control
ToggleButtonSize	Sets the size of the ToggleButton

Methods for TWebJQXResponsivePanel

Refresh	Performs a refresh of the control. If the width of the parent element has changed the ToggleButton is hidden or displayed based on the width of the parent element
---------	--

Events for TWebJQXResponsivePanel

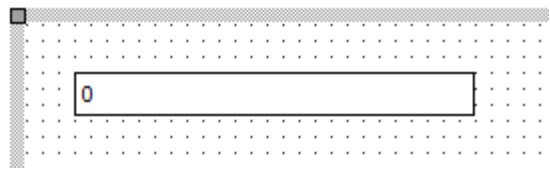

OnClose	Event triggered when the panel is closed by clicking the ToggleButton
OnCollapse	Event triggered when the window (or parent element) width is higher than CollapseBreakPoint and the ToggleButton is hidden
OnExpand	Event triggered when the window (or parent element) width is lower than CollapseBreakPoint and the ToggleButton is

	displayed
OnOpen	Event triggered when the panel is opened by clicking the ToggleButton

TWebJQXSlider

Description

Below is a list of the most important properties methods and events for TWebJQXSlider. Represents a control that lets the user select from a range of values by moving a thumb along a track.

 <p>Designtime</p>	 <p>Runtime</p>
--	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebJQXSlider

ButtonsPosition	Sets the position of the buttons. Options are Both (Left and Right), Left, Right. Only available when ShowRangeSlider is False
-----------------	--

Maximum	Sets the maximum value of the slider
MaximumValue	Sets the maximum selected value when ShowRangeSlider is True
Minimum	Sets the minimum value of the slider
MinimumValue	Sets the minimum selected value when ShowRangeSlider is True
MinorTicksFrequency	Sets the frequency of the minor ticks
MinorTicksSize	Sets the size of the minor ticks
Mode	Sets the mode of the slider. Options are Default or Fixed. If fixed the
ShowButtons	Sets if the buttons are displayed. Only available when ShowRangeSlider is False
ShowMinorTicks	Sets if the minor ticks are displayed along the slider
ShowRange	Sets if the slider range background is displayed
ShowRangeSlider	Sets if the slider is displayed as a range slider and has 2 thumbs. This allows to select a minimum and maximum value
ShowTicks	Sets if the ticks are displayed along the slider
Step	Set the slider's increment and decrement step when the thumb is moved
Template	Sets the template used to display the control. Options are Default, Primary, Success, Warning, Danger, Info
Theme	Sets the name of the theme that is used to display the control
TickSize	Sets the size of the ticks
TicksPosition	Sets the position of the ticks. Options are Both (above and below the slider), Bottom or Top
Value	Sets the default value

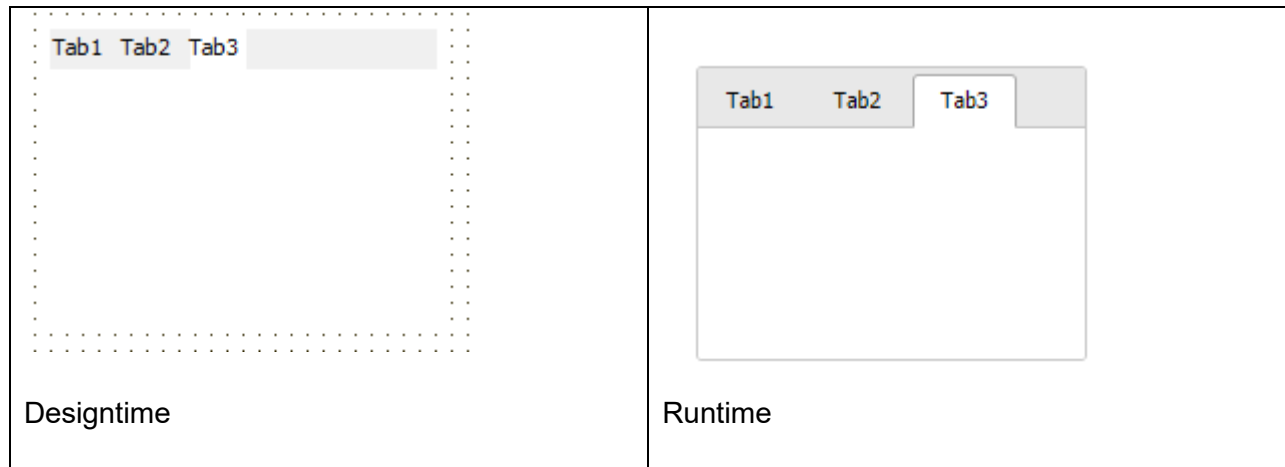
Events for TWebJQXSlider

OnChange	Event triggered when the value is changed
----------	---

TWebJQXTabs

Description

Below is a list of the most important properties methods and events for TWebJQXTabs. TWebJQXTabs is similar to a VCL TPageControl.



HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebJQXTabs

Collapsible	Sets if the tab is collapsible by clicking the selected tab
EnableHover	Sets if a hover effect is displayed when hovering a tab with the mouse cursor
EnableScrollAnimation	Sets if animation is used when scrolling through the tabs
Position	Sets the position of the tabs row. Options are Top and Bottom

Reorder	Sets if the tabs can be reordered with drag and drop
ScrollPosition	Sets the position of the scrollbar arrows. Options are Left, Right and Both
ScrollStep	Sets the distance in pixels that is scrolled with the scroll arrows
SelectionTracker	Sets if an animated effect is displayed when switching between tabs
TabIndex	Sets the index of the active tab
ToggleMode	Sets the method used to select a tab. Options are Click, DoubleClick and MouseEnter
Theme	Sets the name of the theme that is used to display the control

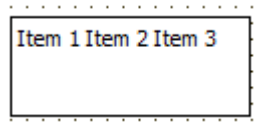

Events for TWebJQXTabs

OnSelected	Event triggered when a new tab is selected
OnTabClick	Event triggered when a tab is clicked

TWebJQXTagCloud

Description

Below is a list of the most important properties methods and events for TWebJQXTagCloud. Represents a control that displays a collection of pre-defined tags. Each tag has a weight value which corresponds with the size of the displayed tag. Tags can also be sorted based on weight or name.

 <p>Designtime</p>	 <p>Runtime</p>
---	---

HTML template tag

The HTML tag the component can be associated with in an HTML template. Assign the ID attribute with a unique value and set the identical value to the ElementID property. Detailed information can be found in the [Use of HTML templates](#) topic.

HTML tag	<DIV ID="UniqueID"></DIV>
ElementID	UniqueID

Properties for TWebJQXTagCloud

DisplayLimit	Sets the maximum number of items displayed
DisplayTopWeighted	When true, the TopWeighted items are prioritized in the list
DisplayMaxValue	Hides items with a value higher than the maximum value
DisplayMinValue	Hides items with a value lower than the minimum value
Items	
Tag	Sets the ID of the Tag
TagLabel	Sets the label text of the Tag
TagName	Sets the name of the Tag
TagValue	Sets the value of the Tag
MaxColor	Sets the text color of the items with the highest value. Together with the MinColor value, tags with will be colored with gradient colors between MinColor and MaxColor
MinColor	Sets the text color of the items with the lowest value. Together with the MaxColor value, tags with will be colored with gradient colors between MinColor and MaxColor
MaxFontSize	Sets the maximum font size of the items with the highest value

MinFontSize	Sets the minimum font size of the items with the lowest value
SortBy	Sets how the items are sorted. Options are: None (original order), Label or Value
SortOrder	Sets if the items are ordered ascending or descending if SortBy is different from None
TextCase	Sets the text case of the items. Options are: Original, UpperCase, LowerCase, FirstUpper, CamelCase
Theme	Sets the name of the theme that is used to display the control

Events for TWebJQXTagCloud

OnClick	Event triggered when a tag is clicked
---------	---------------------------------------

Connecting to data

As Delphi developers we are used to frameworks and components to take the chore out of using databases. Ever since Delphi 1, database handling was abstracted by the TDataSet & TDataSource. Wouldn't it be nice (and mainly productive as this is what is important after all) if this exact abstraction model allowed us to create web applications consuming data? Exactly that goal is what we wanted to achieve with TMS WEB Core, only technically under the hood things are RADically different from the implementation of Delphi 1 like datasets and datasources. So, with TMS WEB Core, you have your DB-aware edit, label, combobox, datepicker etc... and these can be hooked up to a datasource and a datafield can be specified. The dataset though is in this case a wrapper component that will under the hood do its work getting data or updating data via the use of REST HTTP calls to microservices exposed on a data server. As our TMS XData product already provided exactly that: exposing your databases via REST HTTP calls, we extended it to have a web XData client component so you can from Delphi, create a web application against an XData client and hook up your DB-aware components to an XData dataset, pretty much the same way as you can for VCL or FMX native client applications.

For the sake of demo purposes, we have created a first sample app with a web client dataset. This web client dataset gets its data in JSON format from a server via a HTTP REST call. This allows to view and edit the data in the web client dataset but won't do updates server side so that it isn't possible to 'fiddle' with the data and break the sample this way.

Here you can see a form editing contact info with several DB-aware controls, including a DB-navigator.

The screenshot displays a web application interface on a grid background. At the top left is a button labeled "Connect to DB". Below it is a "WebDBNavigator1" control. To the left of the main form area are labels for "Species No:", "Category:", "Common Name:", "Species Name:", "Length cm:", "Length In:", and "Notes:". Each label is followed by a corresponding input control: "WebDBEdit1" for Species No, Category, Common Name, and Species Name; a text box containing "0" for Length cm; "WebDBLabel1" for Length In; and "WebDBMemo1" for Notes. On the right side of the form, there are three data-related controls: "WebClientDataSource1" (with a database icon), "WebClientConnection1" (with a connection icon), and "WebClientDataSet1" (with a dataset icon). At the bottom, a light gray box contains an information icon and the text: "This demo shows a web client dataset connected to DB controls. The web client dataset gets the information from an Client server but for demo purposes all editing in the dataset is local in the web client only!"

When the dataset is connected to the server, the DB-aware controls display and can edit the data.

← → ↻ 🏠 ⓘ www.tmssoftware.biz/tmsweb/demos/tmsweb_dataset/ ☆

TMS WEB Core: Using the web client dataset locally

Connect to DB

I< < > >I ▲ ✓ + - ✕

Species No: 90090

Category: Scorpionfish

Common Name: Firefish

Species Name: Pterois volitans

Length cm: 38

Length In: 14.9606299212598

Notes:

Also known as the turkeyfish. Inhabits reef caves and crevices. The firefish is usually stationary during the day, but feeds actively at night. Favorite foods are crustaceans.

The dorsal spines of the firefish are needle-like and contain venom. They can inflict an extremely painful wound.

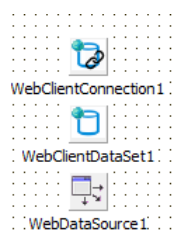
Edibility is poor.

ⓘ

This demo shows a web client dataset connected to DB controls. The web client dataset gets the information from an Client server but for demo purposes all editing in the dataset is local in the web client only!

For viewing data, TMS WEB Core comes with following built-in components:
TWebClientConnection, TWebClientDataSet, TWebDataSource.

Drop the components on the form and assign the WebClientConnection instance to WebClientDataSet.Connection and assign the WebClientDataSet to WebDataSource.



The data will be retrieved via a HTTP GET request in JSON format. To fill the client dataset, it is expected that the JSON consists of a JSON array or a JSON array under a specific node in the

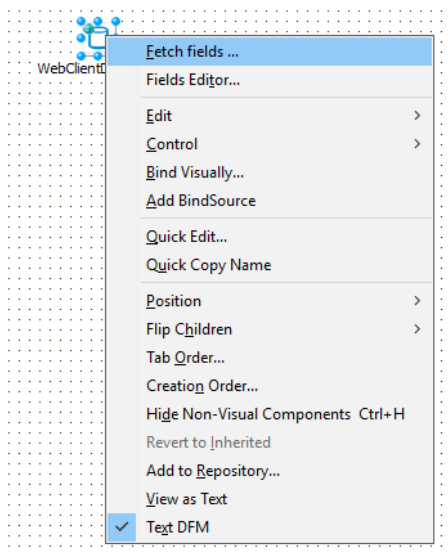
JSON HTTP response.

Specify the URL where the HTTP GET retrieves the JSON data via WebClientConnection.URI.

When the JSON array is under a specific node, specify this with

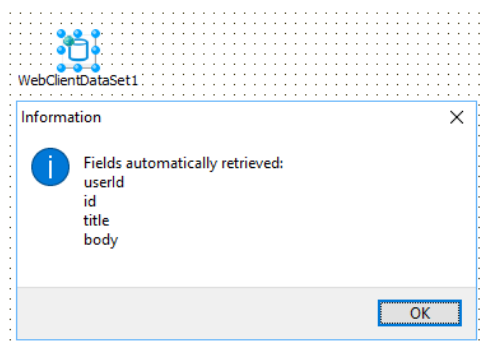
WebClientConnection.DataNode: string;

As JSON as such does not come with meta-data, it will be needed to setup the DB fields expected in the JSON array. Add these as new fields to the dataset via the “Fields Editor” or select “Fetch fields” from the design-time editor of the WebClientDataSet:

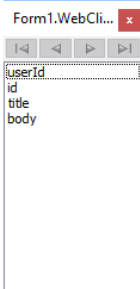


When a WebClientConnection is assigned to the WebClientDataSet and an URL is specified, the IDE will perform a HTTP request and interpret the retrieved JSON and add the DB fields found.

For example, for this sample JSON data at URL: <https://jsonplaceholder.typicode.com/posts> the result is:



After this, we can simply choose from the dataset fields editor “Add all fields” and all fields available in the JSON data will be available for our DB-aware controls:



After setting `WebClientConnection.Active = true`, the `WebClientConnection` performs a HTTP GET request on the URL to fetch the JSON data. This is an asynchronous process. When this is ready, the `OnAfterConnect` event is triggered. When this event is triggered, all data was loaded into the connected `WebClientDataSet` and the data is ready for use. When `WebClientConnection.AutoOpenDataSet = true`, the `WebClientConnection` will automatically open the dataset after this, making it ready to put data in connected DB-aware controls. A typical flow to connect to data, fetch it and then using the dataset directly from code is:

```
procedure TForm1.WebButton1Click(Sender: TObject);
begin
    // start the asynchronous process to perform a HTTP GET request to retrieve the data
    WebClientConnection1.Active := true;
end;

procedure TForm1.WebClientConnection1AfterConnect(Sender: TObject);
begin
    // Data was retrieved in OnAfterConnect, dataset was automatically opened by the
    // WebClientConnection and ready for use
    WebClientDataSet1.First;

    while not WebClientDataSet1.Eof do
    begin
        WebListbox1.Items.Add(WebClientDataSet1.FieldByName('email').AsString);
        WebClientDataSet1.Next;
    end;
end;
```

Connecting to a TMS XData based server is one possible way to hook up to databases. Please refer to the TMS XData documentation for information how you can connect to TMS XData exposed databases from a TMS WEB Core application.

You can implement your own interfaces to a database server via REST HTTP calls and over-time we plan to create and offer connectors to such server as Embarcadero RAD server, Google Cloud datastore and several others...

Using WebSockets

TMS WEB Core promises easy, fast and RAD component based web application development. For fast, real-time updates on a web page with light-weight server-communications, WebSockets are an ideal mechanism.

That is why TMS WEB Core also comes with a WebSocket client:



This is a non-visual component that makes it very easy to start using WebSocket based communication. Drop this component on the form, configure the WebSocket hostname & port and call `WebSocketClient.Connect`. When a connection is established, the `OnConnect` event is triggered. From the moment of connection, data sent by the WebSocket server is received via the event `OnDataReceived`. The signature of this event is:

```
procedure OnDataReceived(Sender: TObject; Origin: string; Data:
TJSObject);
```

Origin is the WebSocket server sending the data and the data itself is sent as a JavaScript Object. This means it can be different types. Sending data is equally easy. Simply call

```
WebSocketClient1.Send(AMessage: String);
```

To create an online chat application using this WebSocket technology takes only a few configurations in the component to configure the WebSocket server and a couple of lines of code. There is the logic that performs the Connect & Disconnect:

```
procedure TWebForm1.Connect;
begin
  if FConnected then
  begin
    WebSocketClient1.Disconnect;
  end
  else
  begin
    if WebEdit1.Text = '' then
      ShowMessage('Please enter a name first')
    else
      WebSocketClient1.Connect;
```

```
    end;  
end;
```

To send a message when connected, we simply send the message as color/sender/message pair via the `WebSocketClient.Send()` function. Each chat user can choose a color and messages from the user are displayed in his selected color:

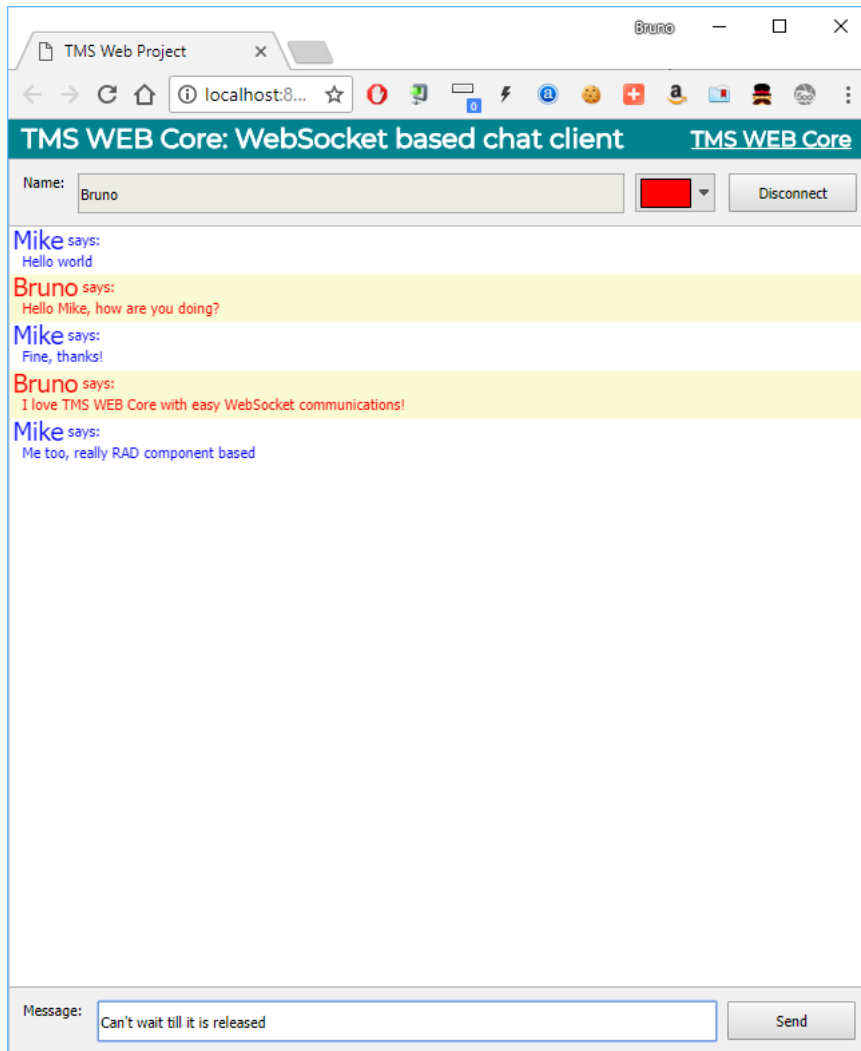
```
procedure TWebForm1.SendMessage;  
var  
    s: string;  
begin  
    if FConnected and (WebEdit2.Text <> '') then  
        begin  
            s := TTMSFNCGraphics.ColorToHTML(TMSFNCColorPicker1.SelectedColor)  
+ '~' + WebEdit1.Text + '~' + WebEdit2.Text;  
            // limit message length  
            s := Copy(s,1,256);  
            WebSocketClient1.Send(s);  
            WebEdit2.Text := '';  
        end;  
    end;  
end;
```

To display the message, we use the web-enabled `TTMSFNCListBox` component from the TMS FNC UI Pack. With this control we can show the received messages in listbox items with banding and some HTML formatting per item to indicate the sender and the message. The message is received via `WebSocketClient.OnDataReceived` as text and therefore we can use `Data.toString` to get the JavaScript object as text:

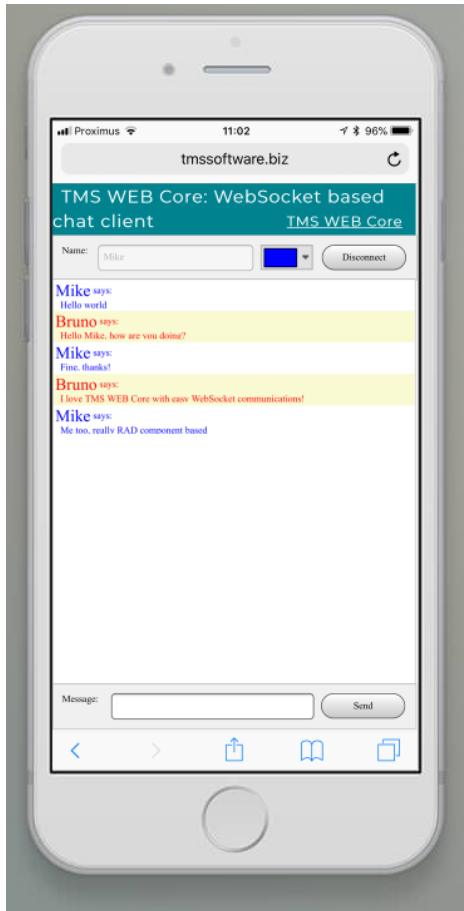
```
procedure TWebForm1.WebSocketClient1DataReceived(Sender: TObject;  
Origin: string;  
Data: TJSObject);  
var  
    it: TTMSFNCListBoxItem;  
    sl: TStringList;  
    s: String;  
    n: string;  
    c: TTMSFNCGraphicsColor;  
    v: string;  
begin  
    it := lst.Items.Add;  
    s := Data.toString;
```

```
sl := TStringList.Create;
try
  TTMSFNCUtils.Split('~', s, sl);
  if sl.Count > 2 then
    begin
      c := TTMSFNCGraphics.HTMLToColor(sl[0]);
      n := '<font size="14">'+sl[1]+'</font>';
      v := sl[2];
      it.Text := n + ' says:<br> ' + v;
      it.TextColor := c;
    end;
  finally
    sl.Free;
  end;
end;
```

There isn't much more to creating a chat application for your TMS WEB Core applications except of course to put a WebSocket server application on your server that can equally be written with Delphi. See the TMS WEB Core demos for a sample WebSocket server service application.



TMS WEB Core chat client application running in the Chrome browser



TMS WEB Core chat client application running in the Safari browser on iPhone

IndexedDB

IndexedDB is a NoSQL database that allows a web application to store anything persistently in the user's browser. Significant amount of structured data can be stored on the client-side including files and blobs. In addition, it provides indexes for fast searching of this data. Each IndexedDB database is unique to a domain or subdomain. It can not be accessed by any other domain. IndexedDB is available in the latest releases of all browsers supported by TMS WEB Core.

TMS WEB Core IndexedDB Library

TMS WEB Core IndexedDB Library provides two ways to create and use IndexedDB databases.

TIndexedDbClientDataset Component

The component TIndexedDbClientDataset makes it easy for a Delphi web application to create and use IndexedDB databases by a familiar syntax of using ClientDataSet. It also allows a seamless integration of an IndexedDB database with data-aware components like TWebDBGrid. All the database operations, including the creation of fields can be done in the standard Delphi way through the TIndexedDbClientDataset component.

Internally, the TIndexedDbClientDataset component uses TIndexedDb class described below to provide this seamless integration thus hiding all the complexity of dealing with asynchronous IndexedDB operations and their responses.

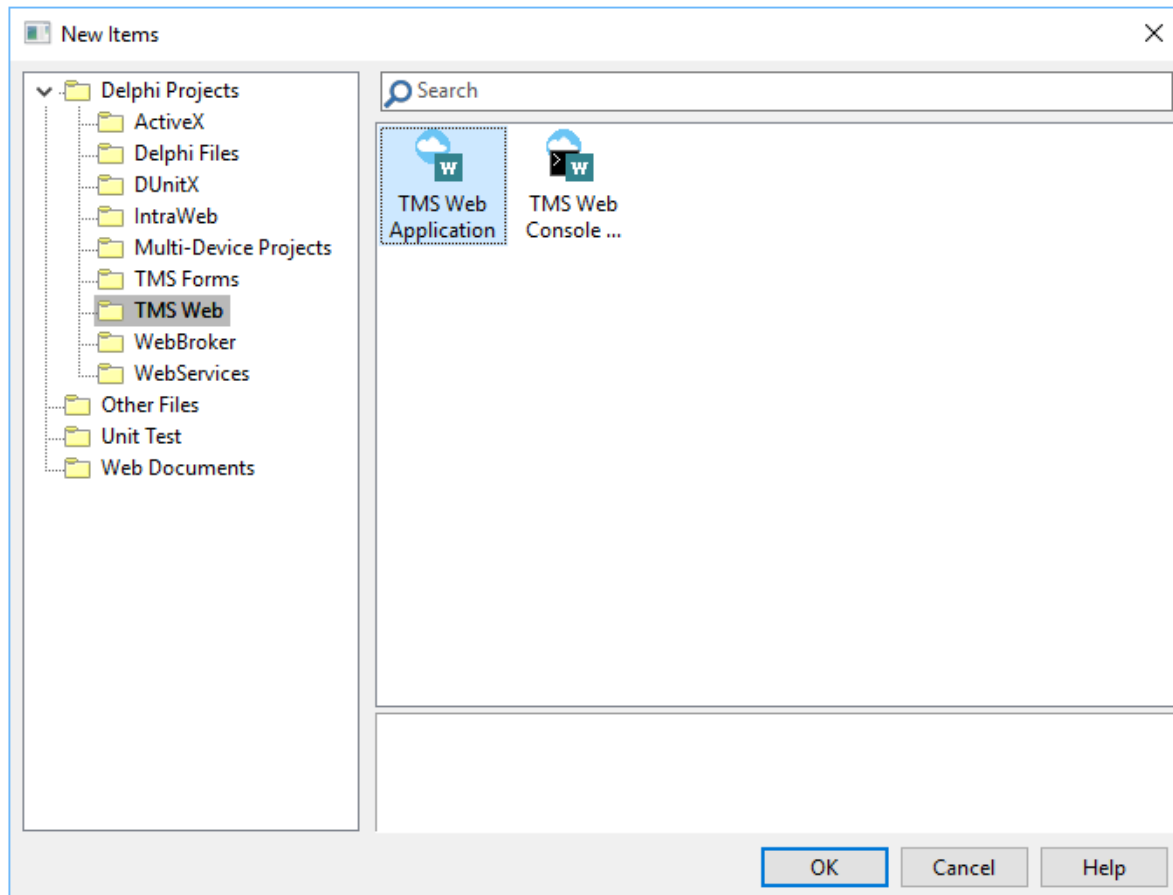
TIndexedDb class

A Delphi web application can also use the class TIndexedDb to directly create and use IndexedDB databases. The original IndexedDB API is low-level and asynchronous. The class TIndexedDb provides easier methods to perform IndexedDB operations where the results are communicated to the specified Delphi event procedures of the application. However, the use of this class needs a basic knowledge of using JavaScript objects, arrays (Pas2Js syntax) and coding Delphi events without using design time aids.

Your first IndexedDB application

Create a TMS web application

Create a standard TMS Web Application in the Delphi IDE by choosing File, New, Other, TMS Web Application. A new web form is created.



Set up the IndexedDB Client Data Set component

Go to the Tool Palette and select the TWebIndexedDbClientDataset component from the “TMS Web Data Access” section and drop it on the web form.

Specify the IndexedDB Database Properties

Set up the properties of the IndexedDB database in the Object Inspector as given below: 1. IDBDatabaseName: NotesDB 2. IDBObjectStoreName: “Notes” 3. IDBKeyFieldName: “id” 4. IDBAutoIncrement: true (default)

This tells the component to use the object store “Notes” in the database “NotesDB.” The primary key field for the object store is specified as “id” which is set up as an auto increment key. The component is smart enough to create the database if it doesn’t exist.

Create the Fields or Properties of each object in the Object Store

The fields of the object store need to be set up in the WebFormCreate event code. In the Object Inspector, double-click on OnCreate event of the Web Form. This creates an event handler procedure WebFormCreate. Type the following code in it that sets up the fields and then makes

the DataSet active.

```
WebIndexedDbClientDataset1.FieldDefs.Clear;  
WebIndexedDbClientDataset1.FieldDefs.Add('id',ftInteger);  
WebIndexedDbClientDataset1.FieldDefs.Add('note',ftString);  
WebIndexedDbClientDataset1.FieldDefs.Add('date',ftDate);  
WebIndexedDbClientDataset1.Active := True;
```

Note that special attention is required when using multiple tables in the same IndexedDB database. Due to the asynchronous nature, create a new table and activating it is not happening synchronously. This implies that when creating and activating multiple (new) tables, this needs to be done after each other. For this reason, the IndexedDbClientDataSet.Init method with anonymous method parameter or OnInitSuccess event is provided. Here the dataset can be easily asynchronously activated after initialization and the initialization of multiple tables can be done after each other.

Here is example code initializing a single IndexedDB database with two different tables used by two different datasets:

```
procedure TMyForm.WebFormCreate(Sender: TObject);  
begin  
    users.FieldDefs.Clear;  
  
    users.FieldDefs.Add('id',ftInteger, 0, true, 3);  
    users.FieldDefs.Add('username',ftString);  
    users.FieldDefs.Add('city',ftString);  
    users.FieldDefs.Add('country',ftString);  
  
    orders.FieldDefs.Clear;  
    orders.FieldDefs.Add('id',ftInteger, 0, true, 3);  
    orders.FieldDefs.Add('product',ftString);  
    orders.FieldDefs.Add('quantity',ftInteger);  
    orders.FieldDefs.Add('price',ftFloat);  
end;  
  
procedure TMyForm.WebFormShow(Sender: TObject);  
begin  
    users.Init(  
        procedure  
        begin  
            orders.Init(  
                procedure  
                begin
```

```

        Userds.Active := true;
        Orderds.Active := true;
    end
)
end
);
end;

```

Add DB-aware components that connect to the DataSet

Now select and drop a TWebDataSource, TWebDBGrid and TWebDBNavigator components on the Web Form.

Set up the DataSource and Data components

Set the DataSource's DataSet property to WebIndexedDbClientDataset1. Then set the DataSource property of the grid and navigator to point to TWebDataSource1.

Set up the Columns of the DBGrid

Do that by clicking on the Columns properties of the DBGrid as shown in the picture.

Set up a New Record event

There is one last thing to do. Since we will be adding New Records or Objects with the DB Navigator, we need to set up the default values of the record. For this, we set up an OnNewRecord event procedure for the IndexedDB Client Data Set in the Object Inspector and type the following code in it.

```

procedure TForm1.NewRecord(DataSet: TDataSet);
begin
    DataSet.FieldByName('note').AsString := 'New Note';
    DataSet.FieldByName('date').AsDateTime := Today;
end;

```

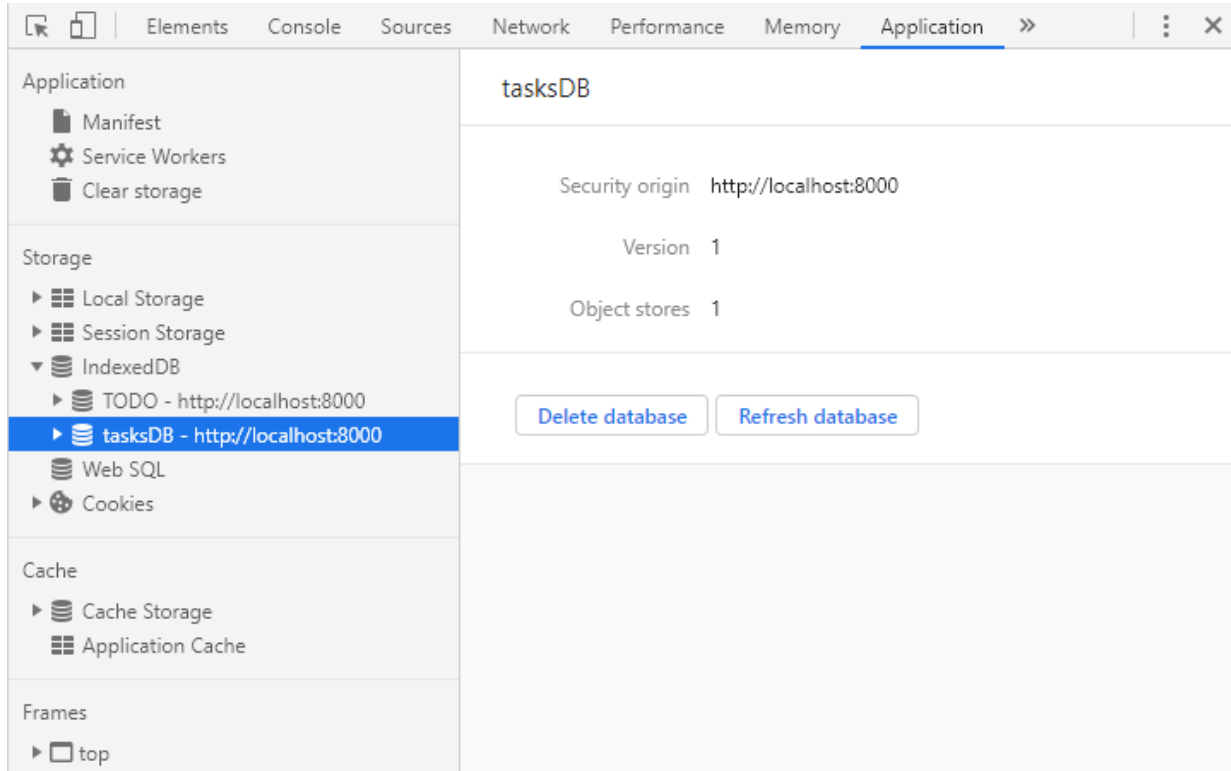
Run the Web Application

Now you can build and run the application. First time, the IndexedDB Client DataSet component will automatically create the database as it doesn't exist. The DB Grid will appear empty as there are no records. Try adding new records with the Navigator and see how it goes.

Managing the IndexedDB Database

In Chrome, start Developer Tools and then select Application. Then see IndexedDB under Storage section. You will see the NotesDB database in it. Here, you can browse through the records and do other operations. For instance, if you change the fields of the database, you can

delete the database itself so that it is recreated with the new fields when you run the application again.



Todo List Demo

Please find this demo in the folder Demos. It shows more advanced usage of using the IndexedDB through the IndexedDbClientDataSet.

Additional features in this Demo

Creating a Permanent Index on a Field

We want to be able to sort on any column of the DB Grid by clicking on the header of the column. So we need to be able to read all the records in the order of that field. For this, we need to create permanent indexes on those fields in IndexedDB. The following code in WebFormCreate event takes care of it.

```
IndexedDBClientDataSet.AddIDBIndex('ByDate', 'due_date');
IndexedDBClientDataSet.AddIDBIndex('ByStatus', 'status');
IndexedDBClientDataSet.AddIDBIndex('ByDescr', 'descr');
IndexedDBClientDataSet.IDBActiveIndex := 'ByDate';
```

The first parameter to AddIDBIndex call is the name that we want to give to an index. The second parameter is the field name. Third parameter is a Boolean specifying isUnique which is OFF by default. Since the fields can contain repeated values, we leave isUnique at default. In order to read the objects in the order of an index, we need to use a code like this:

```
IndexedDBClientDataSet.IDBActiveIndex := 'ByDate';
IndexedDBClientDataSet.IDBIndexDescending := False;
....
IndexedDBClientDataSet.Refresh;
```

The property IDBActiveIndex specifies the objects to be read in the order of 'ByDate' index. Further, the IDBIndexDescending specifies whether the order is Descending or not. The Demo uses this kind of code on the Column Click event of the DB Grid to rekiad it in the desired order. The actual reload is done by the Refresh call.

This Demo also shows an example of connecting Data components like CheckBox or a Memo to the database so that those fields can be edited in the current record. After editing, a call to Update from the update button takes care of committing the changes to the IndexedDB. Similarly the Demo has examples of Inserting a new record and Deleting the current record by respective calls.

TWebIndexedDbClientDataSet

Description

The component TIndexedDbClientDataset makes it easy for a Delphi web application to create and use IndexedDB databases by a familiar syntax of using ClientDataSet. It allows a seamless integration of an IndexedDB database with data-aware components like TWebDBGrid. All the database operations, including the creation of fields can be done in the standard Delphi way through the TIndexedDbClientDataset component.

All you need to do is specify the IndexedDB database properties and add the fielddefs by code in a standard Delphi syntax. Then connect a DataSource and Data components to it and it starts working. It even creates the database if it doesn't exist.

Below is a list of the most important properties and methods of TWebIndexedDbClientDataSet component.

Properties of TWebIndexedDbClientDataSet

Property	Description
Active	Set this to True to activate the ClientDataSet. The IndexedDB database specified by IDB* properties is automatically created if it doesn't exist.
IDBActiveIndex	Set the name of the index to be made active.

	Once this is done, on next Refresh or Active, the ClientDataSet loads the objects in the order of the active index.
IDBAutoIncrement	Set a True or False value to indicate that the primary key field is auto incremented (default True)
IDBDatabaseName	Set the name of the database
IDBIndexDescending	Set a True value to indicate descending order of the active index
IDBKeyFieldName	Set the name of the primary key field
IDBObjectStoreName	Set the name of the ObjectStore in the database
OnIDBError	This is an event property to get notified of any errors. The event can be set up at design time in Object Inspector by double-clicking on it.
OnIDBAfterUpdate	This event is triggered after an asynchronous IndexedDB operation was successfully executed.

Methods of TWebIndexedDbClientDataSet

Only methods specific to IndexedDB are listed. Other methods from the base ClientDataSet class continue to work as before.

Refresh

```
procedure Refresh;
```

Refresh reloads all the objects from the IndexedDB database. If an IDBActiveIndex has been specified, the objects are loaded in the order of that index. In addition, the current record pointer is restored after the Reload.

AddIDBIndex

Use AddIDBIndex to add one or more permanent indexes to the IndexedDB database. Make these calls before you make the component active. The index will be added only if the database does not exist and the component creates it. The call is ignored for an existing database.

```
procedure AddIDBIndex(
    anIndexName: String;
    fields: String;
    isUnique: Boolean=False);
```

Where

- anIndexName - the name to be given to the index
- fields - the field name on which index is to be built. To specify more than one field names, separate the names with semicolons
- isUnique - Specify whether the field value is unique in each object (row)

Notes:

- The call is ignored for an existing database.
- Use the properties IDBActiveIndex and IDBIndexDescending to activate an index for data loading.

TIndexedDb (Advanced Use)

A Delphi web application can use the class TIndexedDb to directly create and use IndexedDB databases.

Using this class needs a knowledge of Pas2Js classes TJSArray, TJObject and their basic syntax in Delphi Pascal. We will try to describe them briefly in “Understanding how the data is stored and retrieved.” But you may skip the rest of this document if you are not interested in a low-level direct access to IndexedDB API.

Description

The original IndexedDB API is low-level and asynchronous. The class TIndexedDb provides a simpler interface with methods to perform IndexedDB operations where the results are communicated to the specified Delphi event procedures of the application.

When several operations are issued simultaneously where the same event procedure of the application gets the response, the application may want to identify the exact operation that was completed. For this purpose, the library provides a way for the application to pass custom data that comes back with the response and helps the application associate a response with an operation.

Below is a list of the most important properties methods and events of TIndexedDb class.

Properties

Property	Description
ActiveIndex	Set the name of the index to be made active. Once this is done, the method GetAllObjsByIndex returns all objects in the order of the active index.
AutoIncrement	Set a True or False value to indicate that the primary key field is auto incremented (default True)
DatabaseName	Read Only property, get the Database Name
IndexDescending	Set a True value to indicate descending order of the active index
KeyFieldName	Read Only property, get the Primary Key Field Name

ObjectStoreName	Read Only property, get the Object Store Name
-----------------	---

Methods and Events of TIndexedDb

The methods and events are listed in their logical order so that they are easier to understand. This is so because IndexedDB only supports asynchronous operation. For each call, there is an event in which the response comes. Hence, describing the events along with the methods makes more sense.

Create and Destroy

These are the standard Delphi methods.

```
constructor Create(AOwner: TComponent);
destructor Destroy;
```

Open or Create an IndexedDB database

Open an IndexedDB database. The database is created if it doesn't exist.

```
procedure Open(
    aDbName: String;
    objectStoreNameToCreate: String;
    KeyFieldName: String = '';
    autoIncrement: Boolean = False;
    sequenceID: Integer = 0);
```

Where

- **aDbName** - the database name
- **objectStoreName** - the object store name
- **KeyFieldName** - the primary key field name. If non-empty, it means an in-line key otherwise an out-of-line key. These terms are described in the section "Understanding how the data is stored and retrieved" below.
- **autoIncrement** - Specify whether the primary key field is AutoIncrement. This works for a new database only. For an existing database, it is ignored and if different a warning is shown
- **sequenceID** - This is an optional ID to be passed to identify the response of this open in case multiple opens are issued simultaneously.

OnResult: Response Event of all methods like Open that perform operations on IndexedDB

Before calling open, you need to setup the OnResult event property to point to an event procedure. The same event gets responses of other operations too.

Format of the event procedure for OnResult:

```
procedure DoOnResult
  (success: Boolean;
   opCode: TIndexedDbOpCode;
   data: JSValue;
   sequenceId: JSValue;
   errorName: String;
   errorMsg: String);
```

where

- **success** - indicates a True or False
- **opCode** - helps identify the operation. Can be one of the following opcodes: opOpen, opAdd, opPut, opDelete, opGet, opGetAllKeys, opGetAllObjs, opGetAllIndexKeys, opGetAllObjsByIndex.
- **data** - The data is sent back in the event for all the operations except for Open, Put or Delete. The data type of the parameter is JSValue. Depending on the operation, you need to cast it to proper data that you are expecting. This is further explained in the description of each operations later.
- **sequenceId** - Id from the original call in order to identify the response in case of multiple operations of the same type issued simultaneously. Note that this is also of the type JSValue. So it is upto the application to send any kind of information to identify an operation. For example, it can even send a JS Object in place of sequence id having more information than just an Id number.
- **errorName** - If you see the documentation of any operation in the original IndexedDB docs, you will find the errors mentioned under an error name. That name is passed here in case you want to code some logic based on a particular error.
- **errorMsg** - error message if success is False

After the success of an open, the application may decide to issue other operations like add, get, etc.

Understanding how the data is stored and retrieved

Before discussing the Add, Get and other procedures that add or get data, we need to understand how the Data is stored and retrieved in IndexedDB.

IndexedDB is a NoSQL, object database. In a relational database, a table stores a collection of rows. In IndexedDB, an ObjectStore stores a collection of JavaScript objects.

A brief introduction to a JavaScript Object

A JavaScript or JS object is a collection of named values. We will call these “properties” in further discussion. If you know OOP terms, a JS object is equivalent to a dictionary or map. You

can have any number of properties representing “fields” of data. Further, there can be nested objects. So the value of a property can be a JS object and so on to any level deep. In TMS Web Core, you can create a JS object with a syntax similar to the following:

```
var
  aDataObj: TJSTObject;
begin
  aDataObj := TJSTObject.New;
  aDataObj['name'] := 'John';
  aDataObj['age'] := 44;
```

The above code creates a JS object with 2 properties.

Objects are stored by unique Primary Key

In IndexedDB, a JS object is stored and accessed by a primary key that is unique. You need to specify a primary key at the time of creating an ObjectStore in IndexedDB. This was the parameter `KeyFieldName` described above for the `Open` procedure. There are 2 types of primary keys in IndexedDb.

In-line key specification

When you pass a non-empty `KeyFieldName` to `Open`, it is an in-line key specification. The term “in-line” means the JS Object itself contains the primary key property by that name.

EXAMPLE: Suppose you pass `KeyFieldName` as ‘id’ to the `Open` procedure that creates the `ObjectStore`. Then it means that when calling an `Add` procedure, you should pass the data as a JS Object that contains a property by the name ‘id’ having a unique key value. You are responsible for putting that property value in the JS Object before passing it to the `Add` procedure.

Exception: The only exception is if you also specified `AutoIncrement` primary key for the `ObjectStore` when creating it with `Open`. In that case, you need not add the key property. IndexedDB fills it after adding the object by using its internal key generator.

Note that the `TIndexedDbClientDataset` component described earlier internally uses an in-line key.

Out-of-line key specification

If you pass an empty string as `KeyFieldName` property to the `Open` procedure, it means the `ObjectStore` is created with an out-of-line key. This means that the JS Object does not have an implicit (in-line) key property. Rather, you are supposed to pass a unique primary key value as a separate parameter (out-of-line) to `Add` procedure when adding data.

What is the use of out-of-line keys? You can have the data as any JS data type. In in-line keys, the data must be a JS Object. In out-of-line keys, the data can be anything, for instance, an

integer or a string. For example, a simple picture application might use an out-of-line primary key as the name of a picture file and the data as the “binary” stream of the picture.

Methods to Add Data

AddData method

```
procedure AddData(  
    data: JSValue;  
    sequenceID: JSValue = 0);
```

Data parameter

- Database created with in-line key and AutoIncrement - Pass a JS Data Object. After the Add completes a key property will be created with the key value generated internally.
- Database created with in-line key and non-AutoIncrement - Pass a JS Data Object that must contain the key as a property having a unique value.
- Database created with out-of-line key and AutoIncrement - Pass any kind of data, even primitive data types are possible.
- Database created with out-of-line key and non-AutoIncrement - You can not use this method in this case. Use PutData method described below.

sequenceID parameter - Optional. Pass any value or object that will help you identify a particular add operation out of many in the OnResult event. Or, you can also pass any data as sequenceID that will help in processing the outcome in OnResult event.

What comes back in OnResult data

The key value that was generated in case of AutoIncrement comes back as data parameter of the event procedure.

PutData method

```
procedure PutData(  
    akey: JSValue;  
    data: JSValue;  
    sequenceID: JSValue = 0);
```

Use this method only if the database was created with out-of-line key specification.

Parameters

- **aKey** - For Add operation, you must pass a unique key value. For Modify, pass the value of an existing key.

- **data** - Pass any kind of data, even primitive data types can be passed.
- **sequenceID** - Optional. Pass any value or object that will help you identify a particular PutData operation out of many in the OnResult event. Or, you can also pass any data as sequenceID that will help in processing the outcome in OnResult event.

Methods to Modify or Update Data

PutData method

There are 2 variations of this method. One is already described above for out-of-line key specification. When you pass an existing key value to that PutData, it acts as a Modify operation.

The second variation is without a key.

```
procedure PutData(  
    data: JSValue;  
    sequenceID: JSValue = 0);
```

Use this method only if the database was created with in-line key specification. In this case, the data must be a JS Data Object and an existing key value must be passed as a key property to modify that object or record.

DeleteData method

```
procedure DeleteData(  
    akey: JSValue;  
    sequenceID: JSValue = 0);
```

Pass the key of the object to be deleted. The sequenceID has same meaning as in earlier methods.

Methods to Get Data

GetData method

```
procedure GetData(  
    akey: JSValue;  
    sequenceID: JSValue = 0);
```

Pass the key of the object to fetch. The object comes back as data in OnResult response event. The sequenceID has same meaning as in earlier methods.

GetKeys method

```
procedure GetKeys(  
    sequenceID: JSValue = 0);
```

The data that comes back in the OnResult response event is a JS Array containing all the keys in natural order. The sequenceID has same meaning as in earlier methods.

GetAllObjs method

```
procedure GetAllObjs(  
    sequenceID: JSValue = 0);
```

The data that comes back in the OnResult response event is a JS Array containing all the data objects or items in natural order. The sequenceID has same meaning as in earlier methods.

Methods to Get Data by an Index

How permanent indexes are created in IndexedDB is described in the next section. Here, you will find a description of the methods that get data in the order of a particular index.

GetIndexData method

```
procedure GetIndexData(  
    indexPropertyName: String;  
    akey: JSValue;  
    sequenceID: JSValue = 0);
```

This is similar to the GetData method described earlier except that you also pass the name of an index to use and pass the key as the field value used in that index to fetch the data.

GetIndexKeys method

```
procedure GetIndexKeys(  
    indexPropertyName: String;  
    sequenceID: JSValue = 0);
```

This is similar to the GetIndexKeys method described earlier except that you also pass the name of an index for which you want to get a list of keys.

GetAllObjsByIndex method

```
procedure GetAllObjsByIndex(  
    sequenceID: JSValue = 0);
```

This method is the most useful that is used by ClientDataSet to load the data. It is similar to GetAllObjs described earlier. But it uses the value of 2 properties of the class to determine the index and the order of the objects returned:

- **ActiveIndex** - Set this to the index name to use for the order of objects. If this is set to empty string (default), the objects are returned in the order of the primary key index.
- **IndexDescending** - Set this to False (default) to get the objects in ascending order. Set to True to get them in Descending order.

Setting up the Indexes

AddIndex

Use AddIndex to add one or more permanent indexes to the IndexedDB database. Make these calls before you open the database. The index will be added only if the database does not exist and hence is created on open. The AddIndex call is ignored for an existing database.

```
procedure AddIndex(  
    anIndexName: String;  
    fields: String;  
    isUnique: Boolean=False);
```

Where

- **anIndexName** - the name to be given to the index
- **fields** - the field name on which index is to be built. To specify more than one field names, separate the names with semicolons
- **isUnique** - Specify whether the field value is unique in each object (row)

Notes:

The call is ignored for an existing database.

Using indexes

This is described earlier under “Methods to Get Data by an Index”. There are two types of methods.

- Methods that require Index Name as a parameter. These are GetIndexData and GetIndexKeys.
- Methods that use the properties ActiveIndex and IndexDescending to work according to an index. Currently there is only one such method, GetAllObjsByIndex. However, these properties are also mapped to the TIndexedDbClientDataset component for easy use of indexes when loading data internally by using GetAllObjsByIndex.

Handling Asynchronous behavior of IndexedDB

If you do a Modify, Delete or Insert, there is a new event that if assigned gets a notification after it completes.

It is `ONIDBAfterUpdate` and the signature is as follows.

```
TKeyId = record
    value: JSValue;
end;

TIDBAfterUpdateEvent = procedure(success: Boolean; opCode:
TIndexedDbOpCode; keyId: TKeyId; errorName, errorMsg: String) of
object;
```

Opcode specifies a modify, insert or delete enum. KeyId is the key of the new or modified record in case you want to use it.

Here is a sample code that adds 5 records on a button press.

```
var
    addNum: Integer = 1;
    countAdded: Integer = 0;

procedure TForm1.btAddMultipleRecordsClick(Sender: TObject);
begin
    countAdded := 0;
    IndexedDBClientDataSet.ONIDBAfterUpdate := DoAfterInsert;
    IndexedDBClientDataSet.Insert;
    IndexedDBClientDataSet.FieldName('descr').AsString := Format('Task
%d', [addNum]);
    IndexedDBClientDataSet.FieldName('status').AsString :=
cbTaskStatus.Text;
    IndexedDBClientDataSet.FieldName('due_date').AsDateTime :=
pickTaskDate.Date;
    IndexedDBClientDataSet.Post;
end;

procedure TForm1.DoAfterInsert(success: Boolean;
```



```
    opCode: TIndexedDbOpCode; keyId: TKeyId; errorName, errorMsg:
string);
begin
    if not Success then
    begin
        ShowMessage('Error: ' + errorMsg);
        IndexedDBClientDataSet.ONIDBAfterUpdate := nil;
        Exit;
    end;
    // Do something with ID added if needed
    Console.log(Format('Id of the new record: %d',
[integer(keyId.value)]));
    Inc(countAdded);
    // Add next record
    if countAdded = 5 then
    begin
        ShowMessage('5 records added successfully.');
```

IndexedDBClientDataSet.ONIDBAfterUpdate := nil;

Exit;

end;

Inc(addNum);

IndexedDBClientDataSet.Insert;

IndexedDBClientDataSet.FieldName('descr').AsString := Format('Task %d', [addNum]);

IndexedDBClientDataSet.FieldName('status').AsString :=

cbTaskStatus.Text;

IndexedDBClientDataSet.FieldName('due_date').AsDateTime :=

pickTaskDate.Date;

IndexedDBClientDataSet.Post;

end;

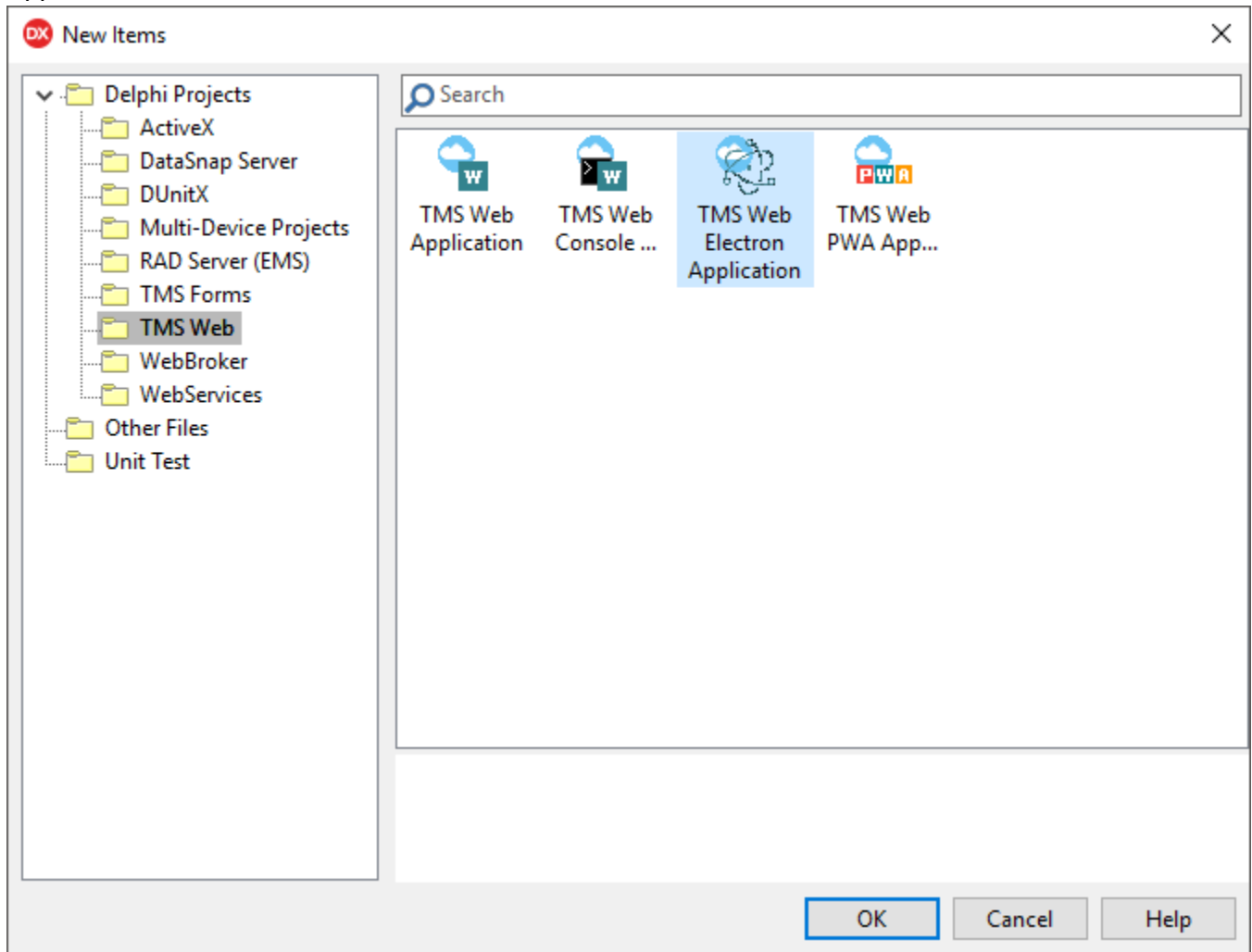
TMS WEB Electron

Electron is an open source library for creating cross-platform desktop applications with HTML, CSS, and JavaScript. Combined with TMS WEB Core, Delphi developers can also create applications for Windows, macOS and Linux by writing the code only once. More information on Electron can be found on the official website: <https://electronjs.org/>

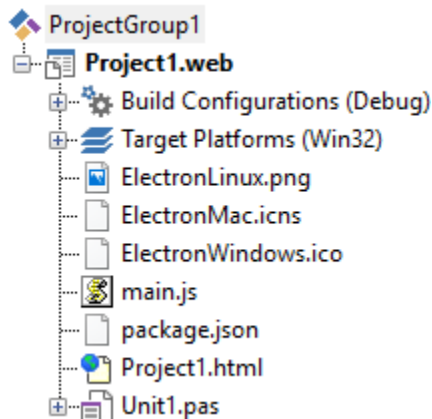
The minimum required Electron version is: 8.0.0. If you already have Electron installed but the version is lower than 8.0.0, then the minimum required version will be installed globally. If a project needs a specific lower version, please install it locally.

Your first TMS Web Electron Application

To create a new Electron application from TMS WEB Core, select the “TMS Web Electron Application” from the wizard:

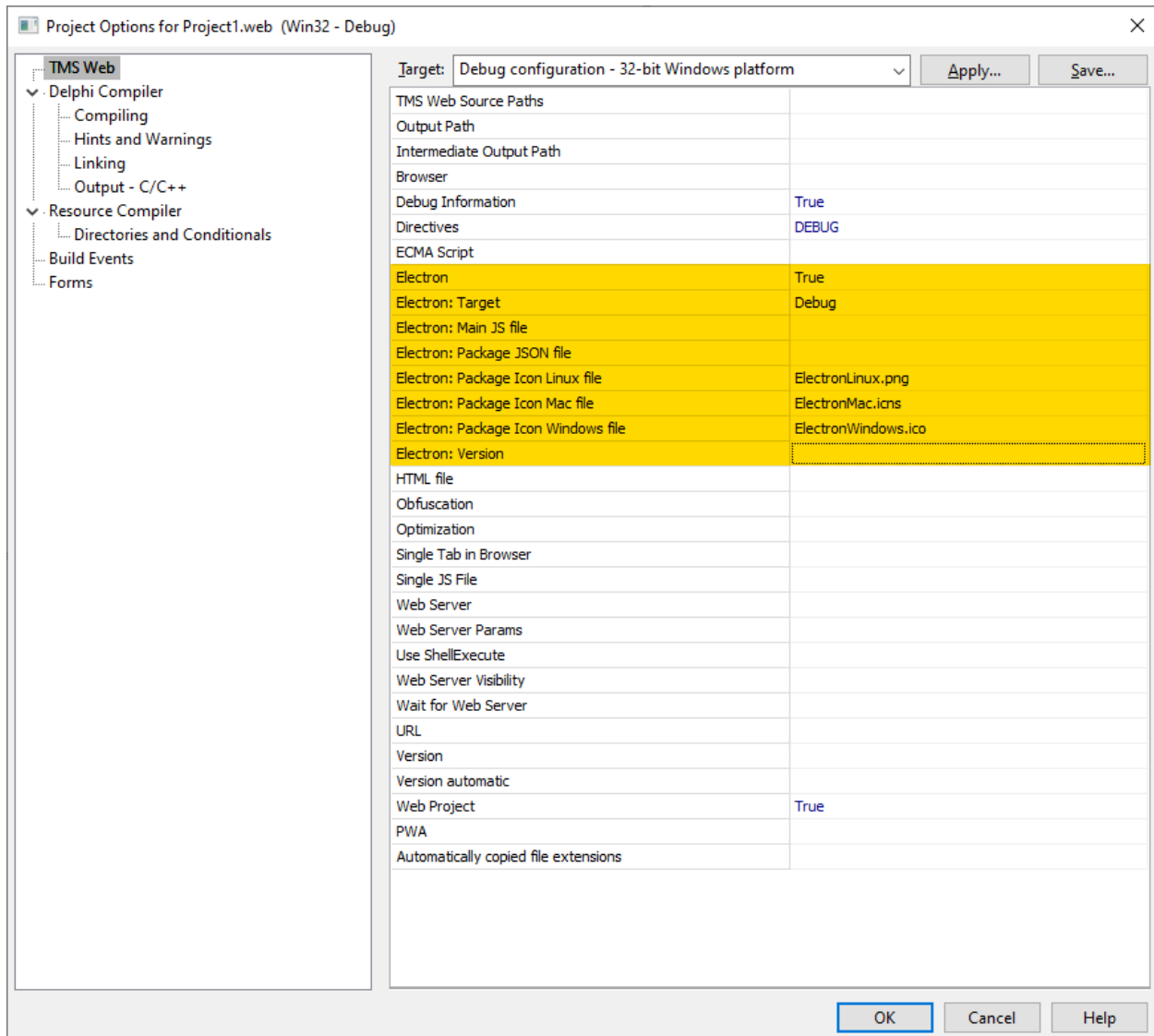


It generates a project similar to a TMS Web PWA Application, but instead of the manifest and serviceworker files, it has generated a main javascript file, a package file and 3 icons for the different platforms:



To every Electron application the package.json is the starting point. This is where the engine will search for the main javascript file that it can run. Both the default generated package.json and main.js are capable of creating, running and packaging an application, but they can be further customized by editing them.

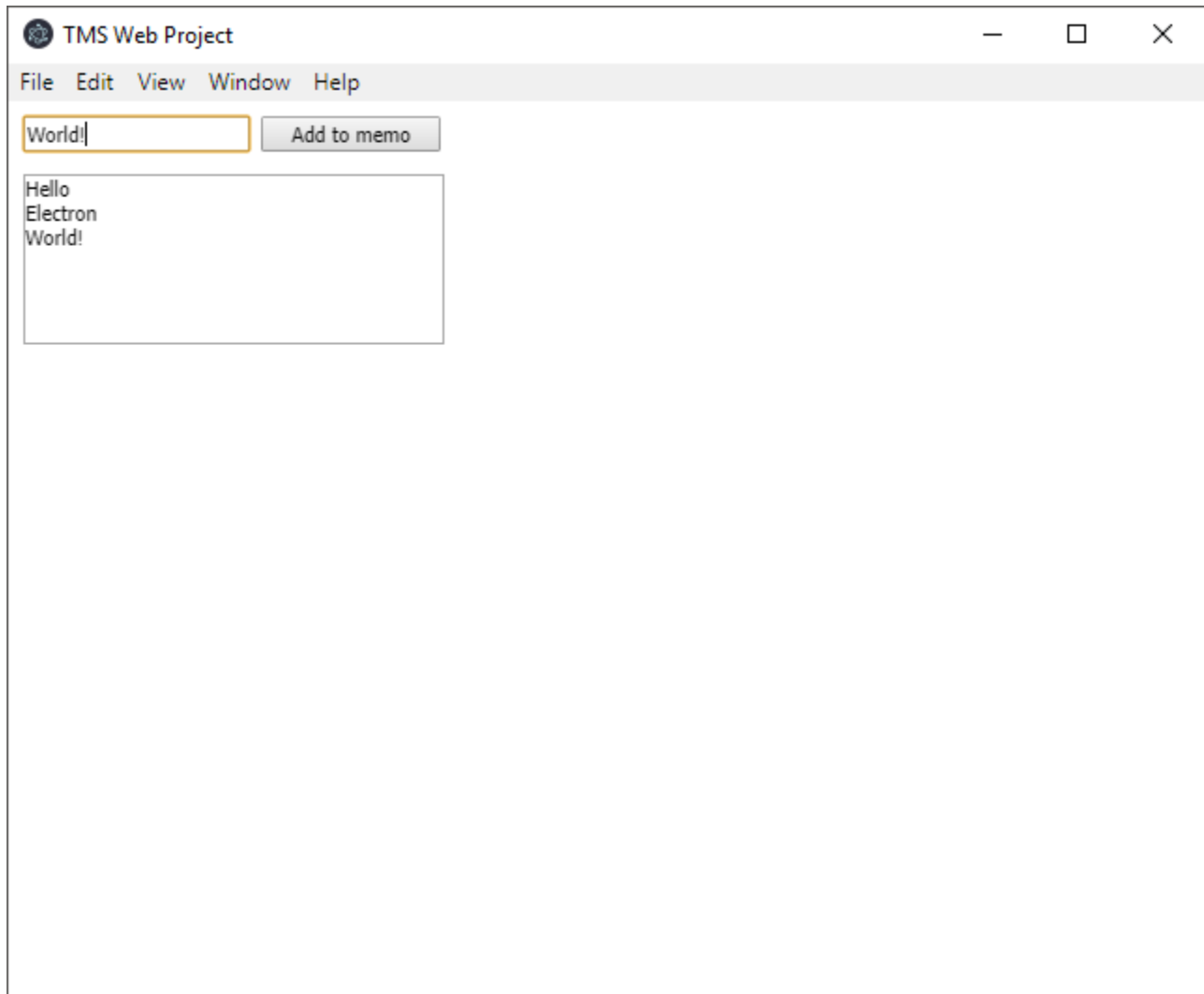
The icon files can be changed through the project options:



When Electron: Version is not specified, the default Electron packager version installed on the machine is used. If you have multiple different Electron packager versions installed on the machine, you can specify here at project level what Electron packager version to use. As there have been quite a few breaking changes between versions before Electron 6 and from Electron 6, a compiler define that you can use in your application code was introduced:

```
{ $IFDEF ELECTRON6UP }
```

You can now develop your application like you would normally do with a TMS Web Application.



Building the application

In Debug mode, pressing F9 will compile the source code, then start up the Electron engine and launch the application.

There are however small differences between the available 6 Build modes.

- **Build-Win32/Build-Win64** will create a packaged 32/64-bit Windows application. If F9 was pressed for building, then it will launch the application after packaging it.
- **Build-Linux32/Build-Linux64** will create a packaged 32/64-bit Linux application. After copying the application to a Linux machine, it can be run.

- **Build-Mac32/Build-Mac64** currently only copies the source files to the Build folder, because it's not possible to create a packaged macOS application on a Windows machine. The application can be created by copying the source code and running [electron-packager](#) on the Mac.

Building on a Mac

If you are unfamiliar with macOS in general and you have no idea how to start with building, you can follow these steps:

Prepare the system by installing [NodeJS](#).

Use npm in the terminal to install [electron-packager](#).

```
//for global installation:  
npm install electron-packager -g
```

```
//for local installation:  
npm install electron-packager
```

Navigate to the folder where the source code is, then from the terminal, run the following command:

```
npm run build-mac64
```

This produced a packaged application that can run on your Mac. It's recommended to sign the application if it's for distribution. More information about that can be found

here: <https://electronjs.org/docs/tutorial/code-signing>

Migrate your application to newer versions

With WEB Core v1.4 we had to introduce changes that are not backwards compatible due to changes in the Electron framework. Below you will find a few steps to help you get started with migrating your application.

Replace the main.js file

This is **required** for every project! The new code depends on the new main.js file. If you wish to work on a project that was created under previous versions, you will first need to replace the main.js. If you modified the main.js then you need to copy your code to the new main.js file as well.

Dialog callbacks

The dialogs now require callbacks if you want to return any results and not just show a message. Here are two code snippets to show you the file opening:

Before WEB Core v1.4:

```
procedure TForm1.WebButton1Click(Sender: TObject);
var
  sl: TElectronStringList;
begin
  if ElectronOpenDialog1.Execute then
  begin
    sl := TElectronStringList.Create;
    sl.LoadFromFile(ElectronOpenDialog1.FileName);
    Meditor.Lines.Assign(sl);
    sl.Free;
  end;
end;
```

From WEB Core v1.4:

```
procedure TForm1.OpenDialogCallback(FileNames: TJSElectronStringDynArray);
var
  sl: TElectronStringList;
begin
  if Length(FileNames) > 0 then
  begin
    sl := TElectronStringList.Create;
    try
      sl.LoadFromFile(FileNames[0]);
      MEditor.Lines.Assign(sl);
    finally
      sl.Free;
    end;
  end;
end;

procedure TForm1.WebButton1Click(Sender: TObject);
begin
  ElectronOpenDialog1.Execute(@OpenDialogCallback);
end;
```

Remove the Sender parameter from TElectronIPCCommunication.OnMessage

The Sender parameter allowed you in the past to send a message directly back to the sender. Unfortunately with the removal of the remote module we don't have access to the BrowserWindow object from the renderer processes anymore.

Remove TElectronIPCMain related codes

TElectronIPCMain had to be removed from our code because it was also depending on the remote module. If you need something from the main process, you need to send a

message from the IPCRenderer and handle the request and response yourself. You can see quite a few examples of this in our source code.

Accessing the Developer Tools

When the application is deployed in Debug mode Electron adds a default menubar to the application if a TElectronMainMenu has not been added to the main form. The Developer Tools can be accessed with View > Toggle Developer Tools or via the Ctrl+Shift+I shortcut. This might be enough if something occasionally needs to be checked.

To force any window to have the Developer Tools opened after the given window is shown, use the following code in the form's OnCreate event:

```
ElectronWindow.OpenDevTools;
```

Drag and drop

Electron provides support for drag and drop functionality. There's a difference between dragging into and dragging out of an application.

In both cases the dragging needs be detected by an event, but at this moment these events for TMS Web components are not yet completed. Until then with some simple javascript, the dragging event detection can be handled.

From desktop to Electron

Dragging something into the application is actually a feature that is supported by HTML5.

Normally the full file path would not be accessible due to a security feature. Electron removes this limitation as Electron applications are meant to run on a desktop using native features of the operating system.

```
procedure TForm1.WebFormCreate(Sender: TObject);
type
  TDropProc = reference to procedure(ElectronFL: TJSElectronFileList);
var
  el: TJSHTMLElement;
  LDropProc: TDropProc;
begin
  el := WebMemo1.ElementHandle;
  LDropProc := @HandleFileDragDrop;
asm
  el.ondragover = (e) => {
    e.preventDefault();
  };

  el.ondrop = (e) => {
    e.preventDefault();
    let efl = e.dataTransfer.files;
    LDropProc(efl);
  };
end;
```



```

    }
end;
end;

procedure TForm1.HandleFileDragDrop(ElectronFL: TJSElectronFileList);
var
    esl: TElectronStringList;
    ef: TJSElectronFile;
begin
    esl := TElectronStringList.Create;
    ef := ElectronFL[0]; //use the first file in case of multiple files
    esl.LoadFromFile(ef.path);
    WebMemo1.Lines.Assign(esl);
    esl.Free;
end;

```

From Electron to desktop

Dragging something out of an Electron application is supported, but the file must already exist on the local file system. If the file does not exist, it's up to the developer to create it on the fly based on the contents from the application. If the file is present, then it takes two steps to drag something:

1. Send a message in the drag start handler via TElectronDragAndDrop with the path to the file that should be dragged out.

```

procedure TForm1.WebFormCreate(Sender: TObject);
type
    TDragProc = reference to procedure;
var
    el: TJSHMTLElement;
    LDragProc: TDragProc;
begin
    el := WebMemo1.ElementHandle;
    LDragProc := @HandleDragStart;
asm
    el.ondragstart = (e) => {
        LDragProc();
    }
end;
end;

procedure TForm1.HandleDragStart;
begin
    ElectronDragAndDrop.StartDrag('/path_to_item');
end;

```

2. Set a dragging icon through TElectronDragAndDrop (for example in the OnCreate event of the form). Setting an empty icon path might work on Windows but not on other platforms.

```
ElectronDragAndDrop.ListenToDrag('/path_to_icon');
```

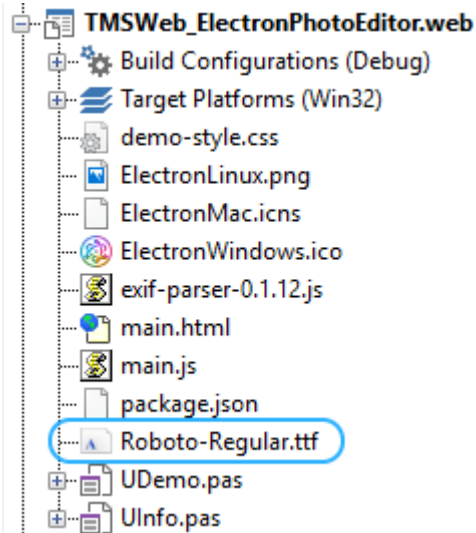
If some code needs to be executed when the dragging is happening, ElectronDragAndDrop has an OnStartDrag event which can be assigned.

```
procedure TForm1.WebFormCreate(Sender: TObject);
  procedure DoDragEvent(Sender: TObject);
  begin
    //Code
  end;
begin
  ElectronDragAndDrop.OnStartDrag := DoDragEvent;
end;
```

Fonts

Sometimes a nice looking application that has been created, does not look the same on another platform, because the font differs. But why is the font not being shown as set? This is due to the fact that different platforms can have different sets of fonts installed. If the used font is not installed, then of course it cannot be used by the application. If this behavior is not desired, it can be fixed by adding a font manually:

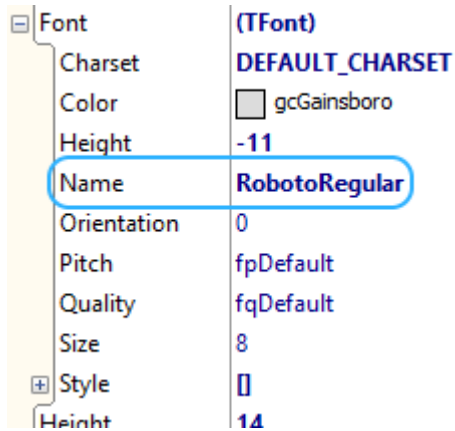
The first step is to add the font file to the project



Add the following code to the project html file

```
<style>
  @font-face {
    font-family: 'RobotoRegular';
    src: url(Roboto-Regular.ttf) format('truetype');
  }
</style>
```

Set the font at any control's Font.Name property.



Set up your project with local databases

Currently we provide support for the following database management systems:

- MySQL: [mysql](#)
- PostgreSQL: [pg](#)

The setup for development (= Debug configuration) and production (= Build-Platform configuration) is identical.

1. Add the correct dependency to the package.json file, and set the version based on your preferences. Example for mysql:

```
"dependencies": {
  "mysql": "^2.18.1"
}
```

2. Build the project without running it. Based on which configuration you picked the output folder will be created (TMSWeb/Debug, TMSWeb/Build-...).

3. Open a commandline prompt / terminal in the output folder and install the module you'd like to use. Keep in mind that if you Clean your project, you'll need to install it again.

```
//mysql node module:
npm install mysql
```

```
//mysql node module with a given version:
npm install mysql@X.Y.Z
```

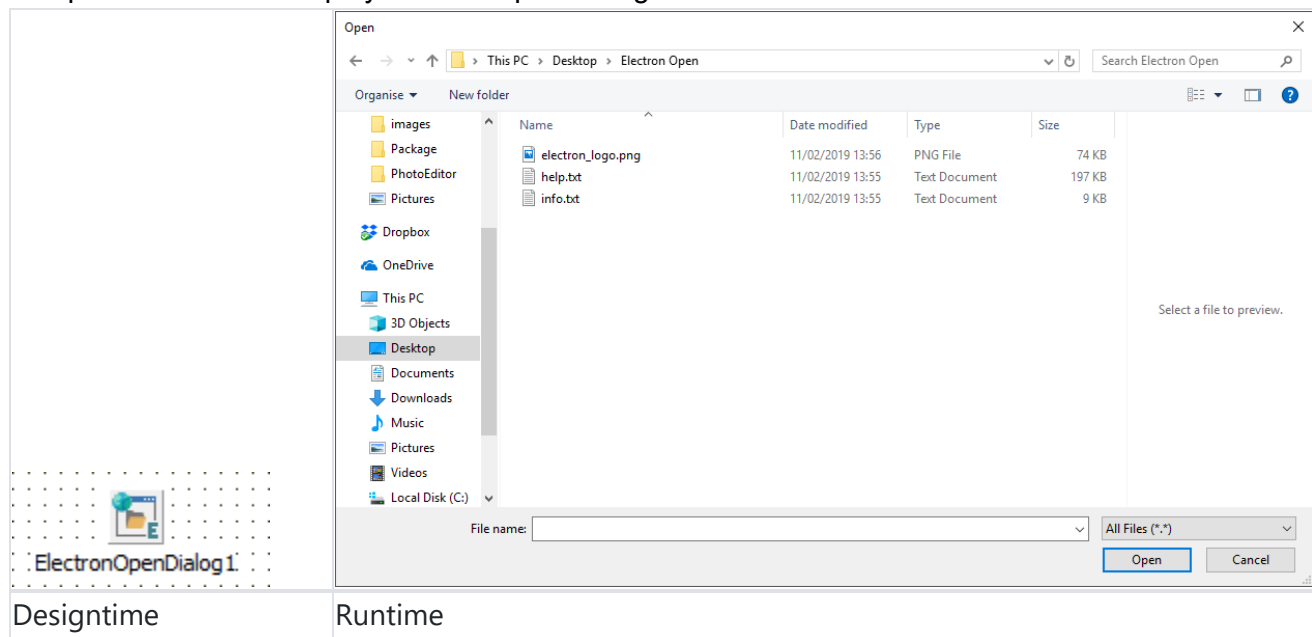
4. Now you can build and run your project again.

Electron Components

TElectronOpenDialog

Description

Below is a list of the most important properties and methods for TElectronOpenDialog. This component allows to display a native open dialog.



Properties for TElectronOpenDialog

Property	Description
ButtonLabel: string	Sets the text that will be shown inside the default “Open” button.
DefaultPath: string	Sets the default path where the dialog is opened.
FileName: string	Returns the filename with full path that has been opened.
Filters: string	Sets the file type filters.
Options	A set of options. On Windows and Linux an open dialog can’t be a file selector and a directory selector at the same time. Choosing both will result in a directory selector.

Property	Description
Title: string	Sets the title of the open window.

Methods for TElectronOpenDialog


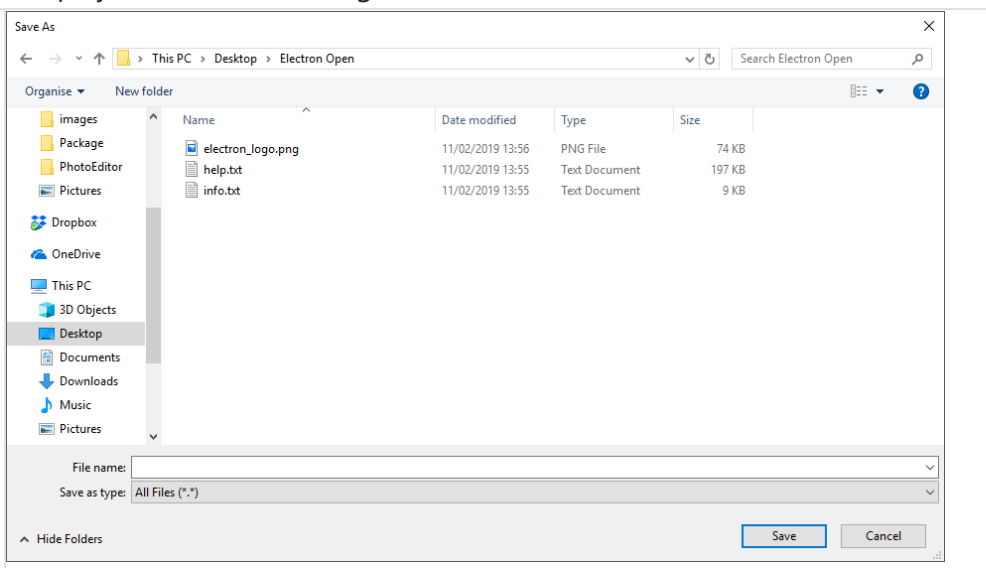
Property	Description
Execute(AProc: TSelectOpenFileCallBack)	Function to show the open dialog. The AProc parameter is a method pointer for a method that is called when the dialog is closed. Any result from the dialog is available through the callback.

See example usage at TElectronStringList: Example 1.

TElectronSaveDialog

Description

Below is a list of the most important properties and methods for TElectronSaveDialog. This component allows to display a native save dialog.

	
Designtime	Runtime

Properties for TElectronSaveDialog

Property	Description
ButtonLabel: string	Sets the text that will be shown inside the default "Save" button.

Property	Description
DefaultPath: string	Sets the default path where the dialog is opened.
FileName: string	Returns the filename with full path that has been saved.
Filters: string	Sets the file type filters.
Title: string	Sets the title of the save window.

Methods for TElectronSaveDialog


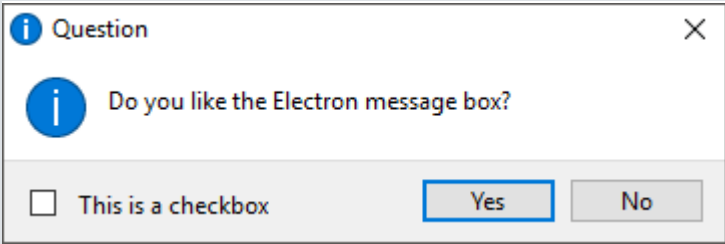
Property	Description
Execute(AProc: TSelectSaveFileCallBack)	Function to show the save dialog. The AProc parameter is a method pointer for a method that is called when the dialog is closed. Any result from the dialog is available through the callback.

See example usage at TElectronStringList: Example 2.

TElectronMessageBox

Description

Below is a list of the most important properties and methods for TElectronMessageBox. This component allows to display a native message dialog.

	
Designtime	Runtime

Properties for TElectronMessageBox

Property	Description
Buttons: TStringList	Sets the buttons
CancelId: Integer	Sets the index of the button to be used to cancel the dialog via the Esc key. By default it's assigned to the first button that has the "Cancel" or "No" label.
CheckboxChecked: Boolean	Sets and returns the checked status of the checkbox.

Property	Description
CheckboxLabel: string	Sets the checkbox text.
DefaultId: Integer	Index of the button from the Buttons list which will be selected by default.
Detail: string	Adds extra information to the Message of the dialog.
DialogType	Sets the type of the dialog. The default value is embNone. On Windows embQuestion has the same icon as embInfo. On macOS embWarning and embError has the same warning icon.
IconPath: string	Sets the path to the icon.
IconURL: string	Base64 encoded string that represents the icon.
Message: string	Sets the content of the dialog.
NoLink: Boolean	On Windows Electron tries to figure out the common buttons from the Buttons list (for example: "Yes", "Cancel"). The rest will be turned into command links.
NormalizeAccessKeys: Boolean	Normalize the keyboard access keys across platforms. Use & in the button label, which then will be converted for each platform accordingly. For example, a button label of Vie&w will be converted to Vie_w on Linux and View on macOS and can be selected via Alt-W on Windows and Linux.
Response: Integer	Returns the index of the button that was clicked.
Title: string	Sets the title of the dialog.

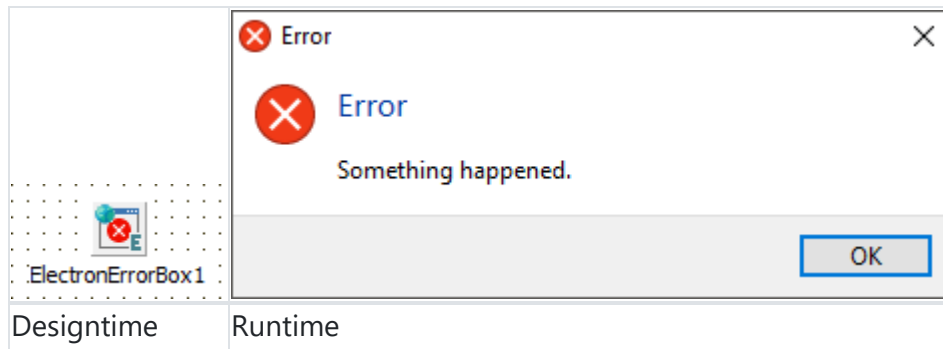
Methods for TElectronMessageBox

Property	Description
Execute(AProc: TSelectMessageBoxCallBack = nil)	Method to show the message dialog. The AProc parameter is a method pointer for a method that is optionally called if assigned when the dialog is closed. Any result from the dialog is available through the callback.

TElectronErrorBox

Description

Below is a list of the most important properties and methods for TElectronErrorBox. This component allows to display a native error dialog.



Properties for TElectronErrorBox

Property	Description
Content: string	Sets the content of the dialog.
Title: string	Sets the title of the dialog.

Methods for TElectronErrorBox

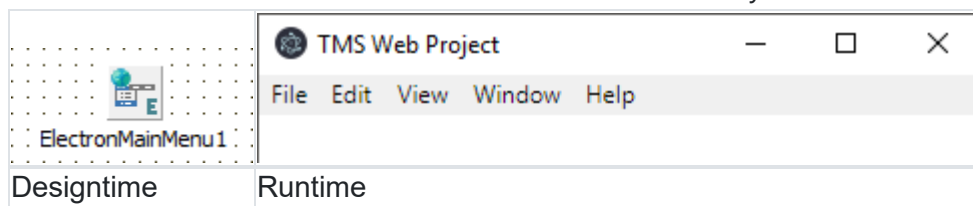
Property	Description
Execute	Function to show the error dialog.

TElectronMainMenu

Description

This component allows to display a native menubar when the application is launched. Creating a TElectronMainMenu and adding TElectronMenuItems to it is similar to VCL's TMainMenu.

TElectronMainMenu should be used in the main form only.



Updating a TElectronMainMenu

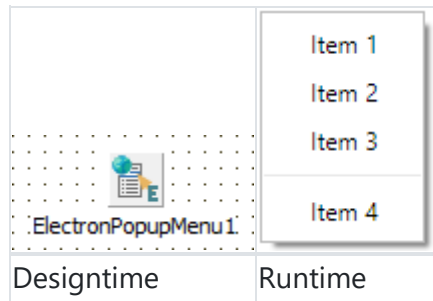
Due to an Electron limitation, it's not possible to update a menu dynamically. This means the menu needs to be recreated and reassigned to the window after each modification. To make this procedure simple, just call the following line after modifying a menu:

```
ElectronMainMenu1.EndUpdate;
```


TElectronPopupMenu

Description

This component allows to display a popup menu. Creating a TElectronPopupMenu and adding TElectronMenuItems to it is similar to VCL's TPopupMenu.



Methods for TElectronPopupMenu

Property	Description
Popup(X, Y: Integer)	Show the popup menu at X and Y coordinates.

Updating a TElectronPopupMenu

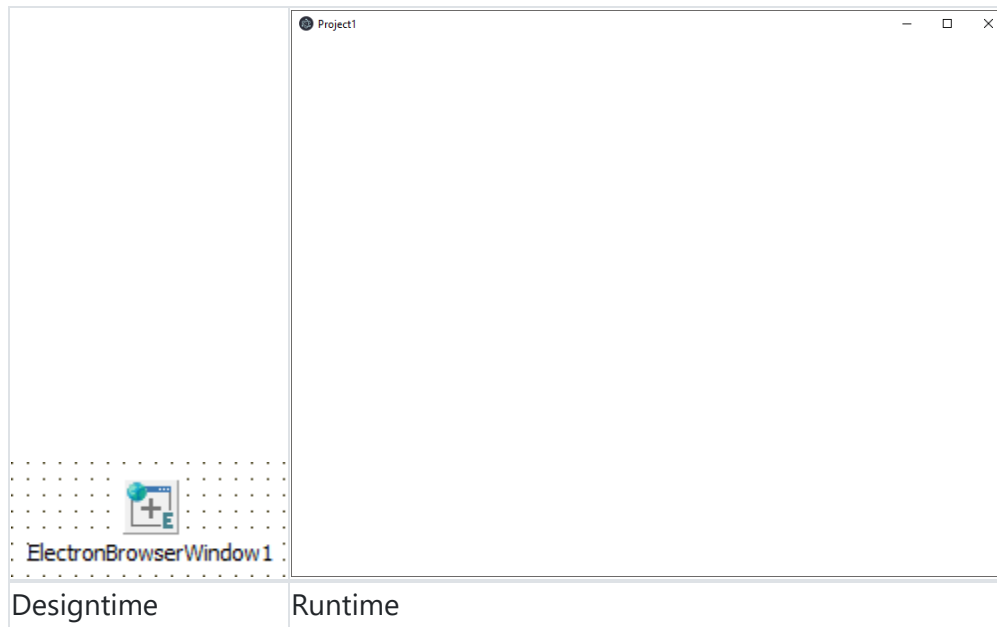
Due to an Electron limitation, it's not possible to update a menu dynamically. This means the menu needs to be recreated and reassigned to the window after each modification. To make this procedure simple, just call the following line after modifying a menu:

```
ElectronPopupMenu1.EndUpdate;
```

TElectronBrowserWindow

Description

Below is a list of the most important properties and methods for TElectronBrowserWindow. This component allows the creation of multiple application windows, which can be linked to forms or other sources.



Properties for TElectronBrowserWindow

Property	Description
FormClass: TFormClass	Sets the form class of the TElectronBrowserWindow.
FullScreen: Boolean	Setting it to true will open the window in fullscreen.
FullScreenable: Boolean	Determines if the window can be set to fullscreen by the user or not.
IconPath: string	Sets the path to the icon.
IconURL: string	Base64 encoded string that represents the icon.
Kiosk: Boolean	Setting it to true will open the window in kiosk mode.
MaxHeight: Integer	Sets the maximum height of the window.
MaxWidth: Integer	Sets the maximum width of the window.
MinHeight: Integer	Sets the minimum height of the window.
MinWidth: Integer	Sets the minimum width of the window.
Resizable: Boolean	Sets the resizability of the window.

Methods for TElectronBrowserWindow

Property	Description
Close	Method to close the window.
ForceClose	Method to force the closing of the window.
Hide	Method to hide the window.

Property	Description
LoadFromURL(URL: string)	Method to load from the given URL. It can be a URL or a path to a local file too.
SendMessage(AMessage: string)	Method to send message to window.
SendMessage(AMessage: JSValue)	Method to send message to the window.
SendMessage(Channel: string; AMessage: string)	Method to send message to a channel.
SendMessage(Channel: string; AMessage: JSValue)	Method to send message to a channel.
Show	Method to show the window.
ShowModal	Method to show the window as a modal It blocks the parent window.

Events for TElectronBrowserWindow

Property	Description
OnActivate	Triggers when the window gains focus.
OnClose	Triggers when the window closes.
OnDeactivate	Triggers when the window loses focus.
OnExitFullScreen	Triggers when the window exits fullscreen mode.
OnFullScreen	Triggers when the window enters fullscreen mode.
OnHide	Triggered when the window gets hidden.
OnMaximize	Triggers when the window is maximized.
OnMessage	Triggers when a message has been sent from the window to the parent window (= the form that contains the TElectronBrowserWindow instance).
OnMinimize	Triggers when the window is minimized.
OnResize	Triggers when the window is resized.
OnShow	Triggers when the window is shown.

Multiple windows using forms

Forms that are added to the project can be used as a source for the page to be shown in the window. In order to achieve this, a few steps have to be made:

1. Whenever a new form is added to the project, it needs to be registered in the initialization section:

initialization

```
RegisterClass(TForm2);
```

2. The form class needs to be assigned to the correct TElectronBrowserWindow instance. To do this, first the unit that contains the form has to be added to the uses list.

For example: We would like to use the TForm2 from Unit2 in Unit1. Then in the uses list of Unit1 add Unit2.

After this, in the form's OnCreate event we can assign the form class to the TElectronBrowserWindow with the code below.

```
ElectronBrowserWindow1.FormClass := TForm2;
```

From now on, whenever ElectronBrowserWindow1.Show is called, it creates the window for us automatically.

Multiple windows using other sources

An HTML file or a link to a website can also be used inside a TElectronBrowserWindow. In this case the only necessary step is to call the URL load method in the OnCreate event of the form.

```
ElectronBrowserWindow1.LoadFromURL('https://www.tmssoftware.com/');
```

or

```
ElectronBrowserWindow1.LoadFromURL('/path_to_html/myFile.html');
```

If the application is targeted for multiple platforms, it's best to use relative paths.

Showing a window

To show the window after its content had been set, simply call Show or ShowModal. The expected behaviour is that showing a modal window will block the parent window until the modal itself gets closed. On macOS this modal window is a sheet that is attached to the parent window, and since it's blocking the parent from closing, it also blocks the whole application from closing. Keep in mind to always provide a way to the user to close the modal window.

TElectronTrayIcon

Description

Below is a list of the most important properties and events for TElectronTrayIcon. This component allows to add a tray icon with an optional popup menu to the application.



Properties for TElectronTrayIcon

Property	Description
IconPath: string	Sets the path to the icon.
IconURL: string	Base64 encoded string that represents the icon.
Menu: TElectronPopupMenu	Sets the menu to be shown when the tray icon is clicked.
ToolTip: string	Sets the tooltip text.

Events for TElectronTrayIcon

Property	Description
OnClick	Triggered when the mouse is clicked on the tray icon.

TElectronIPCCommunication

Description

Below is a list of the most important properties, methods and events for TElectronTrayIcon. This component allows the communication between windows.

Properties for TElectronIPCCommunication

Property	Description
Channel: string	Sets the channel the IPCRenderer is going to listen to. If left blank, it listens to a default channel that is also used by TElectronBrowserWindow.SendMessage(AMessage: string).

Methods for TElectronIPCCommunication

Property	Description
Send(Channel: string; AMessage: JSValue)	Sends a message to the specified channel.
Send(Channel: string; AMessage: string)	Sends a message to the specified channel.
SendToParent(AMessage: JSValue)	Send a message to the parent window.

Events for TElectronIPCCommunication

Property	Description
OnReceive	Triggers when a message has been received on the default channel. If the Channel property is set, then it triggers when a message arrives on that channel.

Send message to parent

Sending a message to the parent window is as easy as dropping a `TElectronIPCCommunication` onto the form and calling

```
ElectronIPCCommunication1.SendToParent('message');
```

Meanwhile on the parent side the sender form's `TElectronBrowserWindow.OnMessage` event will trigger when messages sent via the `SendToParent` method are arriving.

Send message to a channel

It's possible to send a message to a defined channel. This way the communication is enabled between every window and not just parent-child windows.

A specific channel can be defined, but it's not required since every `TElectronIPCCommunication` is listening to a default channel if no Channel is given.

Sending a message to another form is as simple as calling:

```
ElectronIPCCommunication1.Send('Form2', 'message');
```

When the Channel property is defined, then the `TElectronIPCCommunication` component is listening for messages that are arriving on that channel. This allows further possibilities:

- **Multiple messages can be distinguished:** Multiple `TElectronIPCCommunication` instances are dropped onto the form, and each of them are listening to a different channel.
- **Multicast:** On multiple forms the `TElectronIPCCommunication` instances are listening for the same channel.

Receiving messages

Whenever a message has arrived to the channel that's being listened by the `TElectronIPCCommunication`, the `OnReceive` event triggers.

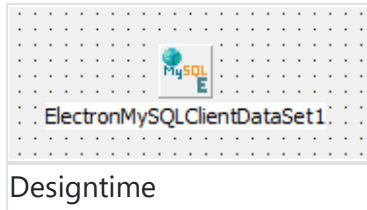
The message it receives is defined as `TReceivedValue`. A message type check can be made before proceeding with the processing of the message:

```
if AMessage.ValueType = jvtString then
begin
    WebMemo1.Lines.Add(AMessage.AsString);
end;
```

The following types are supported: String, Boolean, Integer, Object (TJSObject), Array (TJSArray) and Float.

TElectronMySQLClientDataSet

Description



The component TElectronMySQLClientDataSet makes it easy for an Electron application to create and use MySQL databases by a familiar syntax of using ClientDataSet. It also allows a seamless integration of a MySQL database with data-aware components like TWebDBGrid. All the database operations can be done in the standard Delphi way through the TElectronMySQLClientDataSet component.

Please follow the steps explained in the "Set up your project with local databases" section of this documentation. After the initial setup, all you need to do is specify the TableName and IndexName properties and add the field definitions either in design time or in code in a standard Delphi syntax. Then connect it to a TElectronMySQLConnection component, connect a DataSource and Data components to it and make the dataset active.

Todo List Demo

You can set up a database for the demo either locally by downloading and installing MySQL or by using an online service. In case of an online service it's better to save your credentials somewhere safe as you might not be able to retrieve them later. After the database has been created you can use the following SQL command to create a table that matches the expected syntax of our Todo List Demo:

```
CREATE TABLE tasks (
  id INT AUTO_INCREMENT PRIMARY KEY,
  status VARCHAR(10),
  descr TEXT,
  due_date DATETIME
);
```

Before you run the demo first you need to install the correct node module in the output folder of the project:

1. Build the project and go to the output folder based on your configuration (Debug, Build-...).
2. Install the mysql node module:
`npm install mysql`
3. Now you'll need to build the project again before running it.

Set your credentials in the UMySQL_TodoList.pas unit:

```
electronMySQLConnection.Host := 'your_host_name';
//this is the default port, replace if yours is different
//electronMySQLConnection.Port := 3306;
electronMySQLConnection.DatabaseName := 'your_db_name';
```

Use your username and password at runtime.

BLOB demo

For detailed setup, refer to the Todo List Demo description.

SQL command to create the table:

```
CREATE TABLE files (
  id INT AUTO_INCREMENT PRIMARY KEY,
  file_name TEXT,
  file LONGBLOB
);
```

Set your credentials in the UMySQL_Blob.pas unit:

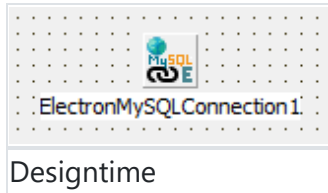
```
ElectronMySQLConnection1.Host := 'your_host';
ElectronMySQLConnection1.DatabaseName := 'your_db_name';
//ElectronMySQLConnection1.Port := 3306;
ElectronMySQLConnection1.User := 'your_user';
ElectronMySQLConnection1.Password := 'your_password';
```

Properties for TElectronMySQLClientDataSet

Property	Description
IndexName	Name of the primary key field of the table.
TableName	Name of the table to connect to.

TElectronMySQLConnection

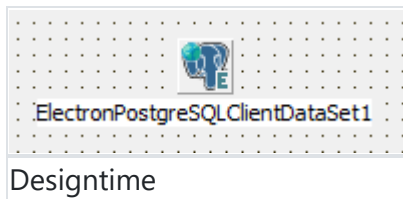
Description



Connection component for the mysql node module. It's required to create a connection. You can also use a single TElectronMySQLConnection component to access multiple tables via multiple TElectronMySQLClientDataSet components.

TElectronPostgreSQLClientDataSet

Description



The component TElectronPostgreSQLClientDataSet makes it easy for an Electron application to create and use PostgreSQL databases by a familiar syntax of using ClientDataSet. It also allows a seamless integration of a PostgreSQL database with data-aware components like TWebDBGrid. All the database operations can be done in the standard Delphi way through the TElectronPostgreSQLClientDataSet component.

Please follow the steps explained in the "Set up your project with local databases" section of this documentation. After the initial setup, all you need to is specify the TableName and IndexName properties and add the field definitions either in design time or in code in a standard Delphi syntax. Then connect it to a TElectronPostgreSQLConnection component, connect a DataSource and Data components to it and make the dataset active.

Todo List Demo

You can set up a database for the demo either locally by downloading and installing PostgreSQL or by using an online service. In case of an online service it's better to save your credentials somewhere safe as you might not be able to retrieve them later. After the database has been created you can use the following SQL command to create a table that matches the expected syntax of our Todo List Demo:

```
CREATE TABLE tasks (  
    id SERIAL PRIMARY KEY,  
    status VARCHAR(10),  
    descr TEXT,  
    due_date DATE  
);
```

Before you run the demo, first you need to install the correct node module in the output folder of the project:

1. Build the project and go to the output folder based on your configuration (Debug, Build-...).

2. Install the pg node module:

```
npm install pg
```

3. Now you'll need to build the project again before running it.

Set your credentials in the UPostgreSQL_TodoList.pas unit:

```
pgConnection.Host := 'your_host_name';  
//this is the default port, replace if yours is different  
//pgConnection.Port := 5432;  
pgConnection.DatabaseName := 'your_db_name';
```

Use your username and password at runtime.

BLOB demo

For detailed setup, refer to the Todo List Demo description.

SQL command to create the table:

```
CREATE TABLE files (  
    id SERIAL PRIMARY KEY,  
    file_name TEXT,  
    file BYTEA  
);
```

Set your credentials in the UPostgreSQL_Blob.pas unit:

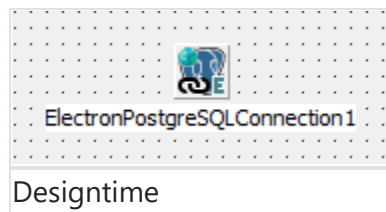
```
ElectronPostgreSQLConnection1.Host := 'your_host';
ElectronPostgreSQLConnection1.DatabaseName := 'your_db_name';
//ElectronPostgreSQLConnection1.Port := 5432;
ElectronPostgreSQLConnection1.User := 'your_user';
ElectronPostgreSQLConnection1.Password := 'your_password';
```

Properties for TElectronPostgreSQLClientDataSet

Property	Description
IndexName	Name of the primary key field of the table.
TableName	Name of the table to connect to.

TElectronPostgreSQLConnection

Description



Connection component for the pg node module. It's required to create a connection. You can also use a single TElectronPostgreSQLConnection component to access multiple tables via multiple TElectronPostgreSQLClientDataSet components.

TElectronFileWatcher

Description

Below is a list of the most important properties for TElectronFileWatcher. This component allows to monitor a list of files for changes.

Properties for TElectronFileWatcher

Property	Description
Files	A container of TElectronFileWatch items

TElectronFileWatch

It has a FileName: string property for the filename and an OnChange event which triggers when the watched file has been modified. The filenames can be relative paths, but please keep in mind

that relative paths might differ on each platform, especially after application packaging. If more than one platform is targeted, then it's recommended to set up the `TElectronFileWatcher` programmatically and use the `ElectronPath` class to retrieve common paths.

TElectronGlobalShortcut

Description

Below is a list of the most important properties and events for `TElectronGlobalShortcut`. This component allows adding listeners for keyboard shortcuts that are recognized even when the application is not in focus. Due to Electron limitation `TElectronGlobalShortcut` should only be used in the main form.

Properties for TElectronGlobalShortcut

Property	Description
Shortcut	Sets the keyboard shortcut.

It's enough to use Ctrl or Cmd only as under the hood it's getting translated to Electron's `CommandOrControl`. On Linux Ctrl key shortcuts sometimes are not working as expected.

Events for TElectronGlobalShortcut

Property	Description
OnShortcut	Triggers when the defined shortcut is pressed on the keyboard.

TElectronStringList

Below is a list of the methods for `TElectronStringList`. This class allows reading from files and writing to files.

Methods for TElectronStringList

Property	Description
LoadFromFile(const FileName: string)	Loads the contents of the file from the local file system into a stringlist.
SaveToFile(const FileName: string)	Writes the contents of the stringlist into a file on the local file system.

Property	Description
string)	system.

Example 1: Open file contents using TElectronStringList and TElectronOpenDialog

```

procedure TForm1.OpenDialogCallback(FileNames: TJSElectronStringDynArray);
var
    sl: TElectronStringList;
begin
    if Length(FileNames) > 0 then
    begin
        sl := TElectronStringList.Create;
        try
            sl.LoadFromFile(FileNames[0]);
            MEditor.Lines.Assign(sl);
        finally
            sl.Free;
        end;
    end;
end;

procedure TForm1.WebButton1Click(Sender: TObject);
begin
    ElectronOpenDialog1.Execute(@OpenDialogCallback);
end;

```

Example 2: Save to file using TElectronStringList and TElectronSaveDialog

```

procedure TForm1.SaveDialogCallback(FileName: string);
var
    sl: TElectronStringList;
begin
    if FileName <> '' then
    begin
        sl := TElectronStringList.Create;
        try
            sl.Assign(WebMemo1.Lines);
            sl.SaveToFile(FileName);
        finally
            sl.Free;
        end;
    end;
end;

procedure TForm1.WebButton2Click(Sender: TObject);
begin
    ElectronSaveDialog1.Execute(@SaveDialogCallback);
end;

```

TElectronBinaryDataStream

Below is a list of the properties and methods for TElectronBinaryDataStream. This class allows reading and writing files using binary data.

Properties for TElectronBinaryDataStream

Property	Description
Base64: string	Set or retrieve the stored data as a Base64 encoded string.
Text: string	Set or retrieve the stored data as a string.

Methods for TElectronBinaryDataStream

Property	Description
LoadFromFile(const FileName: string)	Method to read data from a file from the local file system.
SaveToFile(const: FileName: string)	Method to save data to a file on the local file system.
AsArrayBuffer: TJSArrayBuffer	Function to return the stored data as a TJSArrayBuffer.

TElectronClipboard

Below is a list of the properties and methods for TElectronClipboard. This class allows reading and writing clipboard data. Instead of creating a TElectronClipboard instance, ElectronClipboard can be used.

Properties of TElectronClipboard

Property	Description
AsHTML: string	Sets or returns the data from the clipboard in HTML format.
AsImageURL: string	Sets or returns the data from the clipboard in image data format.
AsRTF: string	Sets or returns the data from the clipboard in RTF format.
AsText: string	Sets or returns the data from the clipboard in plain text (string) format.
FormatCount: Integer	Returns the number of formats that are available in the clipboard at that moment.
Formats	A list of the available formats.

Methods of TElectronClipboard

Property	Description
Clear	Method to clear the data from the clipboard.
HasFormat(Format: string): Boolean	Returns true if the clipboard has data available in the given format.

TElectronShell

Below is a list of available methods for TElectronShell. This class allows invoking native functionalities of the operating system. Instead of creating a TElectronShell instance, ElectronShell can be used.

Methods for TElectronShell

Property	Description
Beep	Method to play the beep sound.
OpenExternal(URL: string): Boolean	Function that tries to open an external link. Returns true if it was successful. Max 2081 characters on Windows, or the function returns false.
OpenItem(FullPath: string)	Function that tries to open an item on the operating system, using the default application. Returns true if it was successful.
MoveItemToTrash(FullPath: string)	Function that tries to move an item to the trash. Returns true if it was successful.
ShowItemInFolder(FullPath: string)	Function that tries to show an item in the folder it is located in. Returns true if it was successful.

TElectronIPCRenderer

Below is a list of the available methods for TElectronIPCRenderer. This class allows listening to channels and sending messages to channels. It's used in the TElectronIPCCommunication component. Instead of creating a TElectronIPCRenderer instance, ElectronIPCRenderer can be used.

Methods for TElectronIPCRenderer

Property	Description
Listen(Channel: string; Listener:	Method to set a listener for a defined channel. The

Property	Description
TIPCEventHandler)	Listener will be invoked everytime a message arrives onto the given channel. Multiple listeners can be assigned to the same channel.
ListenOnce(Channel: string; Listener: TIPCEventHandler)	Method to set a listener for a defined channel that will be executed only once - when the first message arrives onto the channel.
RemoveAllListeners(Channel: string)	Method to remove all listeners on the defined channel.
Send(Channel: string; JSObject: JSValue)	Method to send a message to the given channel.
Send(Channel: string; AMessage: string)	Method to send a message to the given channel.

The listeners (= callbacks) passed to the main process will persist until the main process garbage-collects them. Therefore it's important to uninstall the listeners using the 'RemoveAllListeners' method. More information on this can be found in the Electron documentation: <https://electronjs.org/docs/api/remote>

TElectronDragAndDrop

Below is a list of available methods for TElectronDragAndDrop. This class allows dragging items out of an Electron application. For detailed information please take a look at the Drag and drop part of the documentation. Instead of creating a TElectronDragAndDrop instance, ElectronDragAndDrop can be used.

Property	Description
ListenToDrag(IconPath: string)	Method to listen for dragging events and set the icon path for the dragging. On Windows an empty path might be accepted, on other platforms it's required to have a valid icon path.
ListenToDrag(Listener: TIPCEventHandler)	Method to listen for dragging events. The icon and additional code needs to be added in the Listener.
StartDrag(PathToItem: string)	Method to set the file path to be dragged out of the application.

TElectronPath

Below is a list of the available methods for TElectronPath. This class allows to retrieve common paths accross all supported platforms. Instead of creating a TElectronPath instance, ElectronPath can be used.

Methods for TElectronPath

Property	Description
GetAppData: string	Funcion to retrieve the path to the per-user application data directory.
GetDesktop: string	Function to retrieve the path to the desktop directory.
GetDocuments: string	Function to retrieve the path to the Documents directory.
GetDownloads: string	Function to retrieve the path to the Download directory.
GetExe: string	Function to retrieve the path to the current executable file.
GetHome: string	Function to retrieve the path to the user's home directory.
GetUserData: string	Function to retrieve the path to the application's configuration files. By default this means appData + the application name.
GetPictures: string	Function to retrieve the path to the Pictures directory.
GetTemp: string	Function to retrieve the path to the temporary directory.
GetVideos: string	Function to retrieve the path to the Videos directory.

TElectronWindow

Below is a list of available methods for TElectronWindow. This class allows to control the page that has been loaded into the window (for example: edit commands). Instead of creating a TElectronWindow instance, ElectronWindow can be used.

Methods for TElectronWindow

Property	Description
CloseApp	Method to invoke an application closing call.
CloseWindow	Method to invoke a window closing call.
CloseDevTools	Method to close the developer tools.
Copy	Method that executes the copy editing command.
CopyImageAt(X, Y: NativeInt)	Method that copies the image at the given position to the clipboard.
Cut	Method that executes the cut editing command.
Delete	Method that executes the delete editing command.
DownloadURL(URL: string)	Method that initiates the download of the resource at the URL without navigating. It will prompt a save dialog.
GetURL: string	Function that retrieves the current URL.
OpenDevTools	Method to open the developer tools.
Paste	Method that executes the paste editing command.
Redo	Method that executes the redo editing command.
Reload	Method for reloading the current window.
Replace(AText: string)	Method that executes the replace editing command.
ReplaceMisspelling(AText: string)	Method that executes the replaceMisspelling editing command.
SelectAll	Method that executes the selectAll editing command.
ToggleDevTools	Method to toggle the developer tools.
Undo	Method that executes the undo editing command.
Unselect	Method that executes the unselect editing command.

Other available methods

Property	Description
CreateBlobFromFile(FileName: string): TJSBlob	Function that creates a TJSBlob from a file using the given file name.
CreateUint8ArrayFromFile(FileName: string): TJSUint8Array	Function that creates a TJSUint8Array from a file using the given file name.
DownloadToFolder(URL, Path: string)	Method to download the resource from the given URL to the given Path, without prompting a saving dialog.

Property	Description
GetElectronFileList(Source: TJSEvent): TJSElectronFileList	Function to be used with the WEB components' drag and drop functionality. From the source it will create an array of TJSElectronFile which equals to TJSHTMLFile with an additional path property.
ShortCutToText(ShortCut: TShortCut): string	Function that creates an Electron accepted shortcut string from a TShortCut.

Custom control development

Under the TMS RADical Web umbrella, TMS WEB Core is the foundation framework. As one of the goals of TMS RADical Web is to bring RAD to web development for Delphi developers, it is only logical that using components to develop web applications is an essential part. While TMS WEB Core already comes with a wide range of components out of the box, having an extensible component framework is a key feature. In this article, we will have a look at building custom components for TMS WEB Core.

TMS WEB Core components can be categorized in roughly 4 types:

- Non-visual components
- Visual controls wrapping a HTML element or hierarchy of HTML elements
- Visual controls using the FNC abstraction layer (that cross-framework, cross-platform and web-enabled)
- Visual controls wrapping jQuery controls

Non-visual components

The good news here is that non-visual components for TMS WEB Core are identical to non-visual components for VCL or FMX applications. In TMS WEB Core, the base classes TComponent & TPersistent are available and new non-visual components can be inherited from these base classes and properties, events, methods can be added. The non-visual component can be added to web forms just like VCL non-visual components can be added to VCL forms. Note though that the standard VCL non-visual components included in Delphi can't be used as-is on web forms. After all, all this code needs to run directly in the browser. But already out of the box, TMS WEB Core offers many equivalents to standard VCL non-visual components like the TWebTimer for example being equivalent for TTimer or a TWebDataSource as equivalent for TDataSource.

There is one key requirement to make your custom non-visual component available on the Delphi component palette when a web project is opened and that is to decorate the component with an attribute TMSWebPlatform (defined in WebLib.Controls.pas):

```
[ComponentPlatforms(TMSWebPlatform)]
TMyWebComponent = class(TComponent)
private
// your private variables & methods here
protected
// your protected methods here
public
// your public methods here
published
// your published properties and events here
end;
```

Visual controls wrapping HTML elements

The architecture for this type of controls is based on writing a Pascal class that wraps the HTML element or element hierarchy. The Pascal wrapper class will do the following:

- create the HTML element(s) in the DOM or bind to existing HTML elements in the HTML file associated with the web form
- bind the HTML element JavaScript events to Pascal class methods
- reflect Pascal class properties on HTML element(s) attributes

To create such component, it can inherit from TCustomControl that already includes much of the required logic. Key virtual methods and essential properties defined in TCustomControl are:

```
TCustomControl = class(TComponent)
protected
function CreateElement: TJSElement; virtual;
function ContainerElement: TJSElement; virtual;
procedure BindElement; virtual;
procedure UpdateElementSize; virtual;
procedure UpdateElementVisual; virtual;
procedure UpdateElementData; virtual;
procedure CreateInitialize; virtual;
published
property ElementID;
property ElementClassName;
end;
```

Override the `CreateElement` function to create the HTML element or HTML element hierarchy needed for the control. This function returns a reference to the parent or container HTML element for the control. If only a single HTML element will be needed in the custom control, this is as simple as:

```
function TMyCustomControl.CreateElement: TJSElement;  
begin  
    Result := document.createElement('DIV');  
end;
```

The parent or container element returned by the `CreateElement` function can be retrieved from other places in the control code via the function `ContainerElement`.

The `CreateElement` function will be called automatically from the base class when the control `ElementID` is empty at the time the control parent is set. When `ElementID` is not empty, the container element is retrieved from the DOM based on `ElementID` value, i.e. the control class will use the HTML element returned by `document.getElementById(ElementID)`.

By default, JavaScript event binding is done on the container element. The base class already binds the JavaScript `onwheel`, `onclick`, `onmousedown`, `onmouseup`, `onmousemove`, `onmouseleave`, `onmouseenter`, `onkeydown`, `onkeyup`, `onkeypress`, `onfocus` and `onblur` events. The base class already maps these JavaScript events on virtual class methods with a signature compatible with VCL, that are easy to override. These are for example the available virtual key and mouse event related methods:

```
TCustomControl = class(TComponent)  
protected  
    procedure MouseUp(Button: TMouseButton; Shift: TShiftState; X,Y:  
Integer); virtual;  
    procedure MouseDown(Button: TMouseButton; Shift: TShiftState; X,Y:  
Integer); virtual;  
    procedure MouseMove(Shift: TShiftState; X,Y: Integer); virtual;  
    procedure DoMouseEnter; virtual;  
    procedure DoMouseLeave; virtual;  
    procedure MouseWheel(Shift: TShiftState; WheelDelta: Integer; var  
Handled: Boolean); virtual;  
    procedure DblClick; virtual;  
    procedure KeyDown(var Key: Word; Shift: TShiftState); virtual;  
    procedure KeyPress(var Key: Char); virtual;  
    procedure KeyUp(var Key: Word; Shift: TShiftState); virtual;  
end;
```

So, from our custom control, all we need to do is override these virtual methods, so it is very similar to writing VCL custom controls.

Three more important virtual methods that will typically be overridden are:

```
procedure UpdateElementSize; virtual;  
procedure UpdateElementVisual; virtual;  
procedure UpdateElementData; virtual;
```

The UpdateElementSize procedure is supposed to do the necessary HTML element attribute changes when the position and/or size of the control changes. The base class TCustomControl will already handle this for the container element Top,Left,Width & Height. (when the control is absolute positioned that is).

The UpdateElementVisual method is the place where typically Pascal class properties that affect the visual appearance of controls, are mapped onto HTML element(s) attributes. The UpdateElementData method is the place where properties that affect data connected to controls is updated in the HTML element. To illustrate this, let's assume our custom control mapping on a HTML DIV element has a color property to set background color of the DIV and a text property for the text in the HTML DIV element. The corresponding code for this is:

```
uses  
  Classes, WEBCLib.Controls, Web;  
TMyCustomControl = class(TCustomControl)  
private  
  FColor: TColor;  
  FText: string;  
  procedure SetColor(const AValue: TColor);  
  procedure SetText(const AValue: string);  
protected  
  function CreateElement: TJSElement; override;  
  procedure UpdateElementVisual; override;  
  procedure UpdateElementData; override;  
published  
  property Color: TColor read FColor write SetColor;  
  property Text: string read FText write SetText;  
end;  
  
function TMyCustomControl.CreateElement: TJSElement;  
begin  
  Result := document.createElement('DIV');  
end;
```

```
procedure TMyCustomControl.SetColor(const AValue: TColor);
begin
    if (AValue <> FColor) then
    begin
        FColor := AValue;
        UpdateElementVisual;
    end;
end;

procedure TMyCustomControl.SetText(const AValue: string);
begin
    if (AValue <> FText) then
    begin
        FText := AValue;
        UpdateElementData;
    end;
end;

procedure TMyCustomControl.UpdateElementVisual;
var
    el: TJSHTML_Element;
begin
    inherited;
    el := TJSHTML_Element(ContainerElement);
    el.style.setProperty('background-color', ColorToHTML(Color));
end;

procedure TMyCustomControl.UpdateElementData;
var
    el: TJSHTML_Element;
begin
    inherited;
    el := TJSHTML_Element(ContainerElement);
    el.innerHTML := Text;
end;
```

Finally, to finish this first basic custom control example, let's add a click handler. As the base class already binds the container element 'onclick', this is as simple as:

```
TMyCustomControl = class(TCustomControl)
published
    property OnClick;
end;
```

For the sake of completeness, let's discuss also how to map control methods on HTML element

JavaScript events. This is done with the HTML element `addEventListener()` method.

Example:

```
TMyCustomControl = class(TCustomControl)
protected
    function HandleDoXXXX(Event: TEventListenerEvent): Boolean; virtual;
    procedure BindEvents; override;
end;

procedure TMyCustomControl.BindEvents;
begin
    inherited;
    Container.addEventListener('xxxx', @HandleDoXXXX);
end;

function TMyCustomControl.HandleDoXXXX(Event: TEventListenerEvent):
Boolean;
begin
    // code to be executed when Javascript event XXXX is executed
    Result := true;
end;
```

Assume the HTML event has a JavaScript event named XXXX, the control class method `HandleDoXXXX` will be called when this JavaScript event is triggered.

Visual controls using the FNC abstraction layer

A second approach to create custom controls for TMS WEB Core is by inheriting from the base class `TTMSFNCCustomControl` defined in the TMS FNC Core. The good news is that by doing so, the control code will work for VCL applications, FMX applications, LCL applications and of course also web applications. Technically, for a web application, an FNC web control will internally create a HTML CANVAS element. It maps all needed JavaScript events on this CANVAS to the control virtual methods and it offers an FNC `TTMSFNCGraphics` Pascal wrapper class to perform the painting of these controls.

To get started, the FNC units we will use are:

- `WEBLib.TMSFNCCustomControl` : defines the base class `TTMSFNCCustomControl`
- `WEBLib.TMSFNCGraphics` : defines the class `TTMSFNCGraphics` for painting
- `WEBLib.TMSFNCTypes` : defines various types used with custom controls
- `WEBLib.TMSFNCGraphicsTypes` : defines various types for handling painting

The class interface can be defined as:

```
TMyFNCCustomControl = class(TTMSFNCCustomControl)
private
    FColor: TColor;
    FText: string;
    FDown: boolean;
    procedure SetColor(const AValue: TColor);
    procedure SetText(const AValue: string);
protected
    procedure HandleMouseDown(Button: TTMSFNCMouseButton; Shift:
TShiftState; X, Y: Single); override;
    procedure HandleMouseUp(Button: TTMSFNCMouseButton; Shift:
TShiftState; X, Y: Single); override;
    procedure HandleKeyPress(var Key: Char; Shift: TShiftState);
override;
    procedure Draw(AGraphics: TTMSFNCGraphics; ARect: TRectF);
override;
published
    property Color: TColor read FColor write SetColor;
    property Text: string read FText write SetText;
end;
```

The implementation for the property setters is:

```
procedure TMyFNCCustomControl.SetColor(const AValue: TColor);
begin
    if (AValue <> FColor) then
    begin
        FColor := AValue;
        Invalidate;
    end;
end;
procedure TMyFNCCustomControl.SetText(const AValue: string);
begin
    if (AValue <> FText) then
    begin
        FText := AValue;
        Invalidate;
    end;
end;
```

To have the control draw itself, all we need to do is override the FNC control Draw() virtual method.

```
procedure TMyFNCCustomControl.Draw(AGraphics: TTMSFNCGraphics; ARect:
TRectF);
begin
    inherited;
    if FDown then
        AGraphics.Fill.Color := gcGray
    else
        AGraphics.Fill.Color := Color;
    AGraphics.DrawRectangle(ARect);
    AGraphics.DrawText(10,10,FText);
end;
```

Let's implement for this basic sample a key event handler that will add the character pressed to the control text and the mouse down that will show the control in a different color.

```
procedure TMyFNCCustomControl.HandleKeyPress(var Key: Char; Shift:
TShiftState);
begin
    Text := Text + Key;
end;
procedure TMyFNCCustomControl.HandleMouseDown(Button:
TTMSFNCMouseButton; Shift: TShiftState; X, Y: Single);
begin
    FDown := true;
    Invalidate;
end;
procedure TMyFNCCustomControl.HandleMouseUp(Button:
TTMSFNCMouseButton; Shift: TShiftState; X, Y: Single);
begin
    FDown := false;
    Invalidate;
end;
```

Visual controls wrapping jQuery controls

Creating a Pascal wrapping class for a jQuery UI control has in fact several similarities with creating a wrapping class for HTML elements as jQuery UI controls are exactly that, a hierarchy of HTML elements. What is a bit different is that typically the jQuery control is created by calling a JavaScript function that creates it. The jQuery object then typically exposes its own events

and our control needs to bind to these events. To facilitate this, the TMS WEB Core framework offers a base class `TjQueryCustomControl`. This class adds the virtual method `InitjQuery()` that is called when the jQuery control needs to be created and the function `GetJQID` function that returns the unique jQuery ID for our control. The jQuery control will by default be hosted in a HTML DIV element. What we learned from wrapping HTML elements, is that the virtual methods `UpdateElementVisual()` / `UpdateElementData()` are what is called when property changes need to be reflected in the control jQuery settings or data.

To create a Pascal wrapper class for a jQuery control, the minimal approach is as such:

```
TmyjQueryControl = class(TjQueryCustomControl)
protected
    procedure InitjQuery; override;
end;
procedure TmyjQueryControl.InitjQuery;
begin
    // create the jQuery control here
end;
```

To show the basic concepts, we demonstrate this with a minimal wrapper for the a nice jQuery progress bar offered here: <https://kimmobrunfeldt.github.io/progressbar.js/>

PROGRESS BAR.JS



Following the docs of this library, to create the jQuery progressbar, we need the following JavaScript code for a half circle progressbar:

```
var bar = new ProgressBar.SemiCircle(div, {options});
```

To update the position of the progressbar, we can use `bar.animate(position);` // with position a value between 0 and 1.

So, our full control code becomes:

```
TjQueryProgressBar = class(TjQueryCustomControl)
private
```

```
FPosition: double;
FBar: TJSElement;
procedure SetPosition(const Value: double);
protected
  procedure InitJQuery; override;
  procedure UpdateElementVisual; override;
published
  property Position: double read FPosition write SetPosition;
end;

{ TjQueryProgressBar }
procedure TJjQueryProgressBar.InitJQuery;
var
  eh: TJSElement;
begin
  eh := ElementHandle;
  asm
    this.FBar = new ProgressBar.SemiCircle(eh, {
      strokeWidth: 6,
      easing: 'easeInOut',
      duration: 1400,
      color: '#FFEA82',
      trailColor: '#eee',
      trailWidth: 1,
      svgStyle: null
    });
  end;
end;
procedure TJjQueryProgressBar.SetPosition(const Value: double);
begin
  if (FPosition <> Value) then
  begin
    FPosition := Value;
    UpdateElementVisual;
  end;
end;
procedure TJjQueryProgressBar.UpdateElementVisual;
begin
  inherited;
  if IsUpdating then
    Exit;
```

```
if not Assigned(FBar) then
    Exit;
asm
    this.FBar.animate(this.FPosition);
end;
end;
```

Note here the asm/end blocks in the code. As for reasons of simplicity, we have not taken the effort to create a Pascal wrapper class for the ProgressBar jQuery object, we need to directly access this jQuery object with JavaScript. It is in the asm/end block in our Pascal code that we can directly write this JavaScript code. This code does not get compiled but is just directly generated inline as-is. As you can see, we map a private variable FBar to the created jQuery object created in the InitjQuery call and from the UpdateElementVisual override, this FBar object is accessed to call its animate() function to update the value. Also noteworthy is that from the asm/end block, we access the instance as "this".

After creating an instance of this control, we can simply add the following code to the button click handler:

```
WebjQueryProgressBar1.Position := 0.5;
```

And the result becomes:



Appendix

Browser locale values

Code	Language	Code	Language	Code	Language	Code	Language	Code	Language
af	Afrikaans	hr	Croatian	el	Greek	pl	Polish	sx	Sutu
sq	Albanian	cs	Czech	gu	Gujurati	pt	Portuguese	sw	Swahili
ar	Arabic (Standard)	da	Danish	ht	Haitian	pt-br	Portuguese (Brazil)	sv	Swedish
ar-dz	Arabic (Algeria)	nl	Dutch (Standard)	he	Hebrew	pa	Punjabi	sv-fi	Swedish (Finland)
ar-bh	Arabic (Bahrain)	nl-be	Dutch (Belgian)	hi	Hindi	pa-in	Punjabi (India)	sv-sv	Swedish (Sweden)
ar-eg	Arabic (Egypt)	en	English	hu	Hungarian	pa-pk	Punjabi (Pakistan)	ta	Tamil
ar-iq	Arabic (Iraq)	en-au	English (Australia)	is	Icelandic	qu	Quechua	tt	Tatar
ar-jo	Arabic (Jordan)	en-bz	English (Belize)	id	Indonesian	rm	Rhaeto-Romanic	te	Teluga
ar-kw	Arabic (Kuwait)	en-ca	English (Canada)	iu	Inuktitut	ro	Romanian	th	Thai
ar-lb	Arabic (Lebanon)	en-ie	English (Ireland)	ga	Irish	ro-mo	Romanian (Moldavia)	tig	Tigre
ar-ly	Arabic (Libya)	en-jm	English (Jamaica)	it	Italian (Standard)	ru	Russian	ts	Tsonga
ar-ma	Arabic (Morocco)	en-nz	English (New Zealand)	it-ch	Italian (Switzerland)	ru-mo	Russian (Moldavia)	tn	Tswana
ar-om	Arabic (Oman)	en-ph	English (Philippines)	ja	Japanese	sz	Sami (Lappish)	tr	Turkish
ar-qa	Arabic (Qatar)	en-za	English (South Africa)	kn	Kannada	sg	Sango	tk	Turkmen
ar-sa	Arabic (Saudi Arabia)	en-tt	English (Trinidad & Tobago)	ks	Kashmiri	sa	Sanskrit	uk	Ukrainian
ar-sy	Arabic (Syria)	en-gb	English (United Kingdom)	kk	Kazakh	sc	Sardinian	hsb	Upper Sorbian
ar-tn	Arabic (Tunisia)	en-us	English (United States)	km	Khmer	gd	Scots Gaelic	ur	Urdu
ar-ae	Arabic (U.A.E.)	en-zw	English (Zimbabwe)	ky	Kirghiz	sd	Sindhi	ve	Venda

Code	Language	Code	Language	Code	Language	Code	Language	Code	Language
ar-ye	Arabic (Yemen)	eo	Esperanto	tlh	Klingon	si	Singhalese	vi	Vietnamese
ar	Aragonese	et	Estonian	ko	Korean	sr	Serbian	vo	Volapuk
hy	Armenian	fo	Faeroese	ko-kp	Korean (North Korea)	sk	Slovak	wa	Walloon
as	Assamese	fa	Farsi	ko-kr	Korean (South Korea)	sl	Slovenian	cy	Welsh
ast	Asturian	fj	Fijian	la	Latin	so	Somani	xh	Xhosa
az	Azerbaijani	fi	Finnish	lv	Latvian	sb	Sorbian	ji	Yiddish
eu	Basque	fr	French (Standard)	lt	Lithuanian	es	Spanish	zu	Zulu
bg	Bulgarian	fr-be	French (Belgium)	lb	Luxembourgish	es-ar	Spanish (Argentina)		
be	Belarusian	fr-ca	French (Canada)	mk	FYRO Macedonian	es-bo	Spanish (Bolivia)		
bn	Bengali	fr-fr	French (France)	ms	Malay	es-cl	Spanish (Chile)		
bs	Bosnian	fr-lu	French (Luxembourg)	ml	Malayalam	es-co	Spanish (Colombia)		
br	Breton	fr-mc	French (Monaco)	mt	Maltese	es-cr	Spanish (Costa Rica)		
bg	Bulgarian	fr-ch	French (Switzerland)	mi	Maori	es-do	Spanish (Dominican Republic)		
my	Burmese	fy	Frisian	mr	Marathi	es-ec	Spanish (Ecuador)		
ca	Catalan	fur	Friulian	mo	Moldavian	es-sv	Spanish (El Salvador)		
ch	Chamorro	gd	Gaelic (Scots)	nv	Navajo	es-gt	Spanish (Guatemala)		
ce	Chechen	gd-ie	Gaelic (Irish)	ng	Ndonga	es-hn	Spanish (Honduras)		
zh	Chinese	gl	Galician	ne	Nepali	es-mx	Spanish (Mexico)		
zh-hk	Chinese (Hong Kong)	ka	Georgian	no	Norwegian	es-ni	Spanish (Nicaragua)		
zh-cn	Chinese (PRC)	de	German (Standard)	nb	Norwegian (Bokmal)	es-pa	Spanish (Panama)		
zh-sg	Chinese (Singapore)	de-at	German (Austria)	nn	Norwegian (Nynorsk)	es-py	Spanish (Paraguay)		
zh-tw	Chinese (Taiwan)	de-de	German (Germany)	oc	Occitan	es-pe	Spanish (Peru)		

Code	Language	Code	Language	Code	Language	Code	Language	Code	Language
cv	Chuvash	de-li	German (Liechtenstein)	or	Oriya	es-pr	Spanish (Puerto Rico)		
co	Corsican	de-lu	German (Luxembourg)	om	Oromo	es-es	Spanish (Spain)		
cr	Cree	de-ch	German (Switzerland)	fa	Persian	es-uy	Spanish (Uruguay)		
				fa-ir	Persian/Iran	es-ve	Spanish (Venezuela)		

TUILanguage

This is the list of possible languages and the suffix used for the HTML filename used when the language is set:

IAfar = 'aa'
 IAbkhazian = 'ab'
 IAvestan = 'ae'
 IAfrikaans = 'af'
 I Akan = 'ak'
 IAmharic = 'am'
 IAragonese = 'an'
 IArabic = 'ar'
 IAssamese = 'as'
 I Avaric = 'av'
 I Aymara = 'ay'
 IAzerbaijani = 'az'
 IBashkir = 'ba'
 IBelarusian = 'be'
 IBulgarian = 'bg'
 IBihari = 'bh'
 IBislama = 'bi'
 IBambara = 'bm'
 IBengali = 'bn'
 ITibetan = 'bo'
 IBreton = 'br'
 IBosnian = 'bd'
 ICatalan = 'ca'
 IChechen = 'ce'

IChamorro = 'ch'
ICorsican = 'co'
ICree = 'cr'
ICzech = 'cs'
IOldSlavic = 'cu'
IChuvash = 'cv'
IWelsh = 'cy'
IDanish = 'da'
IGerman = 'de'
IDivehi = 'dv'
IDzongkha = 'dz'
IEwe = 'ee'
IEnglish = 'en'
IEsperanto = 'eo'
ISpanish = 'es'
IEstonian = 'et'
IBasque = 'eu'
IPersian = 'fa'
IFulah = 'ff'
IFinnish = 'fi'
IFijian = 'fj'
IFaroese = 'fo'
IFrench = 'fr'
IWesternFrisian = 'fy'
IIrish = 'ga'
IGaelic = 'gd'
IGalician = 'gl'
IGuarani = 'gn'
IGujarati = 'gu'
IManx = 'gv'
IHausa = 'ha'
IHebrew = 'he'
IHindi = 'hi'
IHiriMotu = 'ho'
ICroatian = 'hr'
IHaitian = 'ht'
IHungarian = 'hu'
IArmenian = 'hy'
IHerero = 'hz'
IInterlingua = 'ia'
IIndonesian = 'id'
IInterlingue = 'ie'

Ilgbo = 'ig'
ISichuanYi = 'ii'
Inupiaq = 'ik'
Ildo = 'id'
Icelandic = 'is'
Italian = 'it'
Inuktitut = 'iu'
IJapanese = 'ja'
IJavanese = 'jv'
IGeorgian = 'ka'
IKongo = 'kg'
IKikuyu = 'ki'
IKuanyama = 'kj'
IKazakh = 'kk'
IKalaallisut = 'kl'
ICentralKhmer = 'km'
IKannada = 'kn'
IKorean = 'ko'
IKanuri = 'kr'
IKashmiri = 'ks'
IKurdish = 'ku'
IKomi = 'kv'
ICornish = 'kw'
IKirghiz = 'ky'
ILatin = 'la'
ILuxembourgish = 'lb'
IGanda = 'lg'
ILimburgan = 'li'
ILingala = 'ln'
ILao = 'lo'
ILithuanian = 'lt'
ILubaKatanga = 'lu'
ILatvian = 'lv'
IMalagasy = 'mg'
IMarshallese = 'mh'
IMaori = 'mi'
IMacedonian = 'mk'
IMalayalam = 'ml'
IMongolian = 'mn'
IMarathi = 'mr'
IMalay = 'ms'
IMaltese = 'mt'

IBurmese = 'my'
INauru = 'na'
INdebele = 'nb'
INepali = 'nd'
INdonga = 'ng'
IDutch = 'nl'
INorwegian = 'no'
INavajo = 'nv'
IChichewa = 'ny'
IOccitan = 'oc'
IOjibwa = 'oj'
IOromo = 'om'
IOriya = 'or'
IOssetian = 'os'
IPanjabi = 'pa'
IPali = 'pi'
IPolish = 'pl'
IPushto = 'ps'
IPortuguese = 'pt'
IQuechua = 'qu'
IRomansh = 'rm'
IRundi = 'rn'
IRomanian = 'ro'
IRussian = 'ru'
IKinyarwanda = 'rw'
ISanskrit = 'sa'
ISardinian = 'sc'
ISindhi = 'sd'
INorthernSami = 'se'
ISango = 'sg'
ISinhala = 'si'
ISlovak = 'sk'
ISlovenian = 'sl'
ISamoan = 'sm'
IShona = 'sn'
ISomali = 'so'
IAlbanian = 'sq'
ISerbian = 'sr'
ISwati = 'ss'
ISotho = 'st'
ISundanese = 'su'
ISwedish = 'sv'

ISwahili = 'sw'
ITamil = 'ta'
ITelugu = 'te'
ITajik = 'tg'
IThai = 'th'
ITigrinya = 'ti'
ITurkmen = 'tk'
ITagalog = 'tl'
ITswana = 'tn'
ITonga = 'to'
ITurkish = 'tr'
ITsonga = 'ts'
ITatar = 'tt'
ITwi = 'tw'
ITahitian = 'ty'
IUighur = 'ug'
IUkrainian = 'uk'
IUrdu = 'ur'
IUzbek = 'uz'
IVenda = 've'
IVietnamese = 'vi'
IWalloon = 'wa'
IWolof = 'wo'
IXhosa = 'xh'
IYiddish = 'yi'
IYoruba = 'yo'
IZhuang = 'za'
IChinese = 'zh'
IZulu = 'zu'